DOCUMENT RESUME

ED 367 286 IR 016 541

AUTHOR	Yoder, Sharon K.
TITLE	Graphing with "LogoWriter."
INSTITUTION	International Society for Technology in Education, Eugene, OR.
REPORT NO	ISBN-1-56484-010-7
PUB DATE	92
NOTE	68p.
AVAILABLE FROM	International Society for Technology in Education, 1787 Agate St., Eugene, OR 97403-1923 ($9.95, 1-9 copies; $7.95 10 or more copies).
PUB TYPE	Guides - Classroom Use - Instructional Materials (For Learner) (051)
EDRS PRICE	MF01 Plus Postage. PC Not Available from EDRS.
DESCRIPTORS	*Computer Assisted Instruction; *Computer Graphics; *Graphs; Integrated Curriculum; Intermediate Grades; Junior High Schools; Middle Schools; Problem Solving; *Programming; Secondary School Mathematics; Secondary School Science
IDENTIFIERS	Examples; Logographs; *LOGO Programing Language; LogoWriter; Turtle Graphics

ABSTRACT

This book discusses four kinds of graphs that are taught in mathematics at the middle school level: pictographs, bar graphs, line graphs, and circle graphs. The chapters on each of these types of graphs contain information such as starting, scaling, drawing, labeling, and finishing the graphs using "LogoWriter." The final chapter of the book provides information on when to use each of the four types of graphs. This book can be used as supplementary material for a graphing unit in a math or science course, or it can be used by itself to teach graphing. The book includes examples for students to follow. The use of the computer as an aid to all aspects of problem solving when creating graphs is emphasized. Prerequisites to using this book involve students' familiarity with "LogoWriter," ability to use turtle graphics, understanding of Turtle-move and Label modes, and ability to use procedures with inputs. In addition to beginning information, the book includes advanced sections which require more programming skill and should be attempted only by knowledgeable Logo programmers. (TMK)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED 367 286

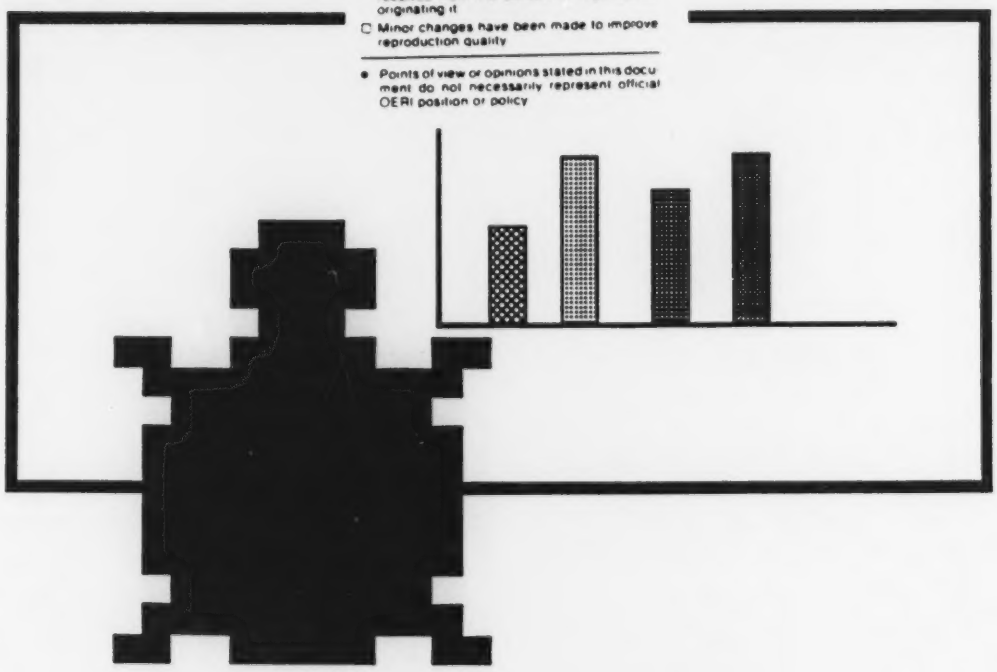
Graphing With *LogoWriter*

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☐ This document has been reproduced as received from the person or organization originating it

☐ Minor changes have been made to improve reproduction quality

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy



1R016541

by Sharon K. Yoder

"PERMISSION TO REPRODUCE THIS MATERIAL IN OTHER THAN PAPER COPY HAS BEEN GRANTED BY
D.Moursund

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)"

BEST COPY AVAILABLE

About the Author

Sharon Yoder has taught mathematics and computer science at the junior high and high school level for 15 years. Her most recent public school experience was as a secondary computer science teacher and a computer coordinator involved in developing system-wide computer curriculum and in planning teacher inservice training.

In addition, she has taught mathematics, computer science, and computer education at a number of universities in northeastern Ohio, including Kent State University, the University of Akron, and Cleveland State University. She worked closely with the College of Education of Cleveland State University in developing their computer programming courses for teachers.

After a year as an Education Specialist for Logo Computer Systems, Inc., Sharon has returned to teaching. She is currently at the University of Oregon, where she teaches computer education courses.

For the past several years, she has conducted workshops and presented papers at conferences nation-wide and has been involved in a number of book publishing projects, including the *Nudges* series, *Introduction to Programming in Logo Using LogoWriter*, *Introduction to Programming in Logo Using Logo PLUS*, and *HyperTalk for Educators: An Introduction*. In addition, she has been a frequent contributor to *The Computing Teacher* and the *Logo Exchange*. She is currently editor of the *Logo Exchange* and *Microsoft Works in Education* newsletter.

Editors: Sharon Yoder and Anita Best

Production: Sharon Yoder

©International Society for Technology in Education, 1992
1787 Agate Street, Eugene, Oregon 97403-1923
Order Desk: 800/336-5191
Phone: 503/346-4414
Fax: 503/346-5890
CompuServe: 70014,2117
Internet: ISTE@Oregon.uoregon.edu

ISBN 1-56484-010-7

Graphing With

LogoWriter

© International Society for Technology in Education, 1992

ISBN 1-56484-010-7

Table of Contents

1. Introduction 3

2. Make a Pictograph 7

   Start Your Pictograph 8
   Scale Your Graph 10
   Draw the Rows of Pictures 12
   Add Labels to Your Graph 14
   *Making Half-shapes 15

3. Make a Bar Graph 19

   Start Your Bar Graph 20
   Scale Your Bars 21
   Place Your Bars 22
   Make an Axis 26
   Scale Your Axes 27
   Label Your Axes 30
   Color Your Bars 32
   *Self-filling Bars 33
   *Debugging Filled Bars 34
   Finish your Graph 36
   *Generalizing Your Bar Graph 37

4. Make a Line Graph 39

   Start Your Line Graph 40
   Create Axes 41
   Label Your Graph 45
   Plot Your Data Points 46
   *Another Way to Plot Points—Use Your Bar Procedure 48
   Finish Your Graph 50

5. Make a Circle Graph 51

   Start Your Circle Graph 52
   Draw a Circle 53
   Draw the Sections of the Graph 56
   Working With Color 57

6. Which Graph Should I Use? 61

Pictographs 61
Bar graphs 61
Line Graphs 61
Circle Graphs 62
Summary 62

Preface
(For the teacher)

This book discusses four kinds of graphs that are taught in math at the middle school level: pictographs, bar graphs, line graphs, and circle graphs. You can use this book as supplementary material for a graphing unit in a math or science course, or you can use it by itself to teach graphing. Students can follow the examples given in the book or use data sets that you give them or that they produce themselves.

To use the book effectively, students need to have a good deal of access to computers. The book emphasizes the use of the computer as an aid to all aspects of problem solving when creating graphs. Thus, students are encouraged to use *LogoWriter* to do computations on data and to scale axes as well as to draw the graphs themselves using the turtle. If you have limited computer access time, then some of the computational activities can be done using a hand held calculator.

Students using this book should be familiar with *LogoWriter*. They need to be able to use turtle graphics with ease, understand Turtle-move and Label modes, and be able to use procedures with inputs. The level of programming required of students depends on the focus of the class. Most of the procedures given in the book can be given to students as "black boxes." The book includes more advanced starred (*) sections. These sections require more programming skill and should be attempted only by more knowledgeable Logo programmers.

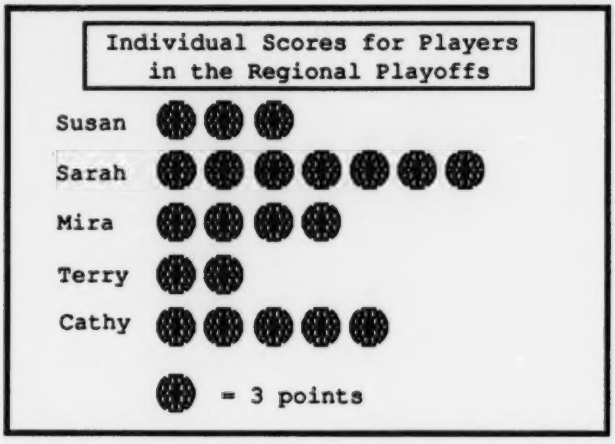
Throughout the book there are occasional boxes.

These boxes contain information designed to help solve problems that might occur as students are working.

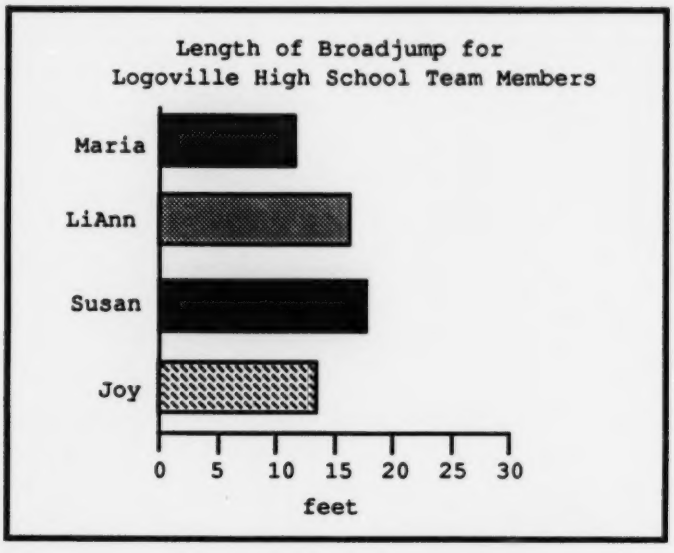
Sometimes these boxes contain information about the differences among the versions of *LogoWriter*; sometimes they contain information specific to the project at hand.

1. Introduction

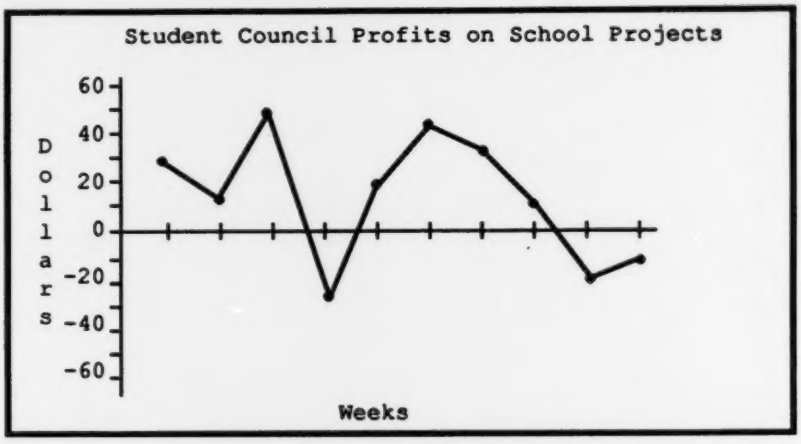
This book is about making graphs. You use graphs to make data easier to understand. Graphs are "pictures" of the data. There are many kinds of graphs. This book will look at pictographs,



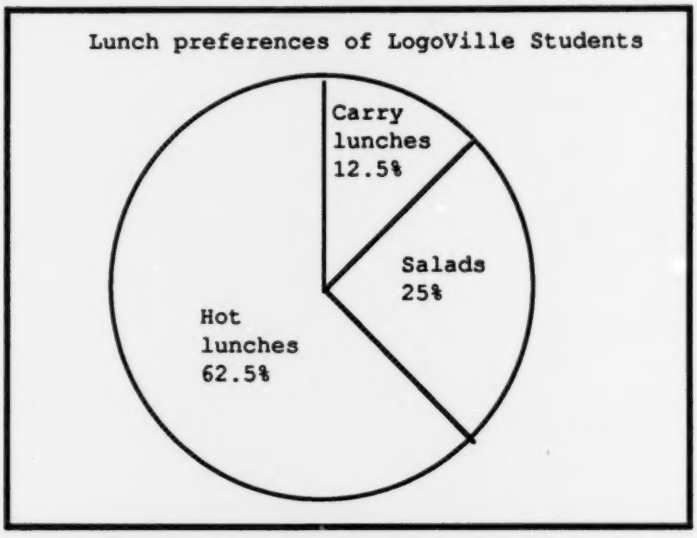
bar graphs,



line graphs,



and circle graphs.



You often make graphs in math or science class. Sometime you make graphs and charts in social studies class. Have you ever used a computer to make a graph? Computers make it easier to create graphs.

In this book, you will learn how to use *LogoWriter* to help you make graphs. Logo can help you to do the arithmetic needed to make graphs. You can use the turtle to draw the graphs. Graphs made with *LogoWriter* are attractive. You can save them. You can change them later. You can print them.

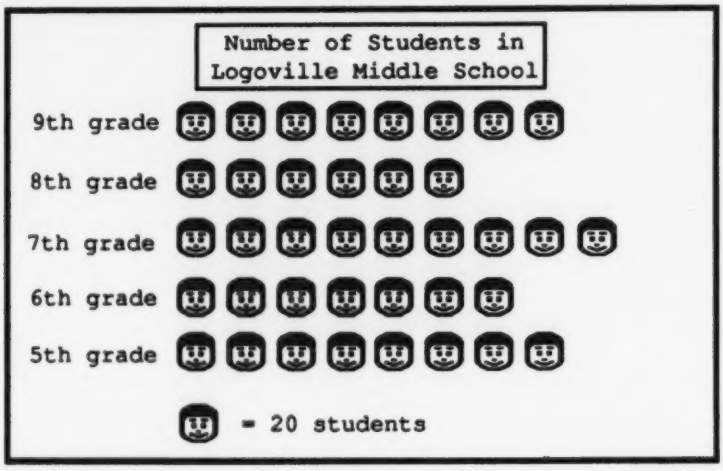
As you work through this book, you will see occasional boxes.

These boxes should help you solve problems as you are working.

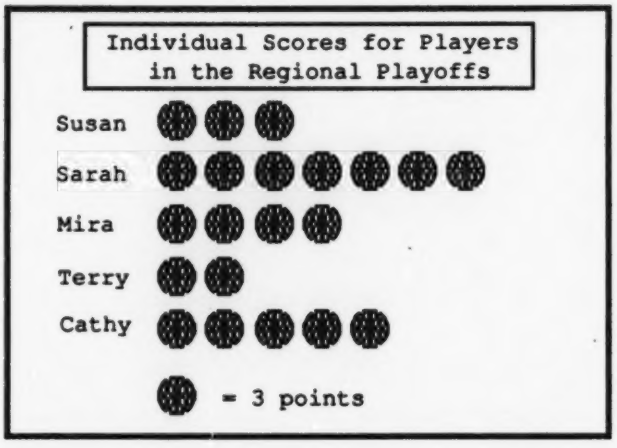
Sometimes these boxes contain information about different versions of *LogoWriter*. Sometimes they contain hints about the project you are working on.

2. Make a Pictograph

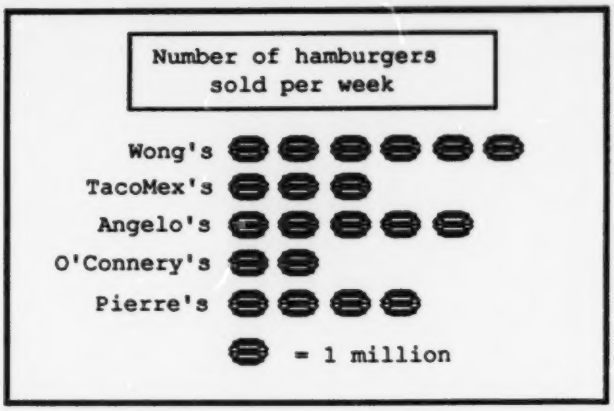
Have you seen charts and graphs in newspapers or magazines that are made up of pictures? These graphs are called *pictographs*.



Usually, the symbol in the graph is a picture that somehow represents the data.



Each symbol in a pictograph often represents a fairly large number.



You can use *LogoWriter* to make graphs such as these.

Start Your Pictograph

The first step in creating a pictograph is collecting some data. You can take a survey and record the results. You can use data from your social studies book. You can do some research in the library. You can do an experiment in science.

When you have collected your data, make a chart to organize it. You can make your chart on paper, or you can use Logo.

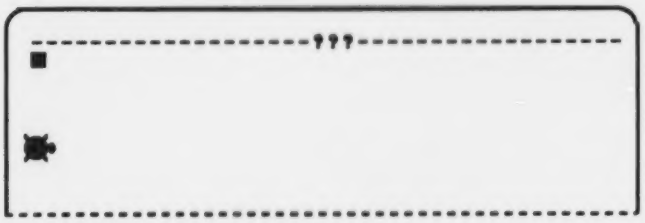
Number of hamburgers sold per week	
Wong's	6,239,000
TacoMex's	2,845,000
Angelo's	5,335,000
O'Connery's	2,007,000
Pierre's	2,700,000

Now you are ready to decide the number that each symbol will represent.

Symbols in pictographs can be used to represent a large number. You can have a stick figure that represents 10 students or a picture of a car that represents the sale of 100 cars. You will make your graph in *LogoWriter*. You must plan your symbol so that the graph fits on the *LogoWriter* page.

The size of the shapes grid is slightly different in each version of *LogoWriter*. Check to see how many full shapes will fit across your screen. Use Turtle-move

to place the turtle on the left edge of your screen. Then turn the turtle to the right.



Set the shape to the solid rectangular shape.

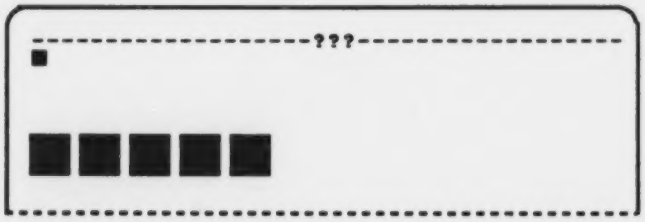
```
setsh 11
```

> Turtle not square shaped? Check your Shapes page. If there is no square shape, create one by completely filling an empty grid.

Then type:

```
repeat 5 [pd stamp pu forward 25]
```

You should see something like this:



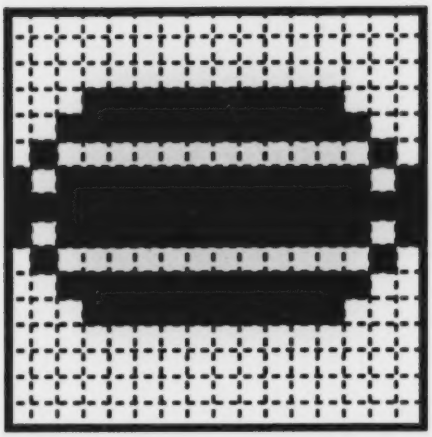
Use the **repeat** instruction to help see how many shapes will fit on your screen.

> No space between your shapes? Make the number after **forward** in the **repeat** instruction bigger.

Remember that you will need some space for the labels on the graph. Usually labels are on the left side. Some pictographs will need more space on the left than others.

Scale Your Graph

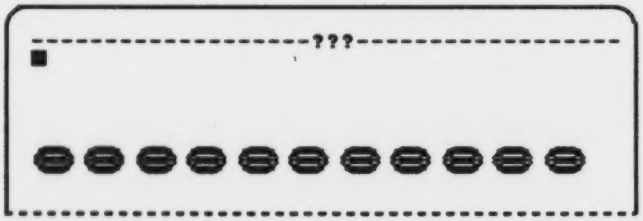
Next, use the Shape editor to create the shape you want to use in your pictograph.



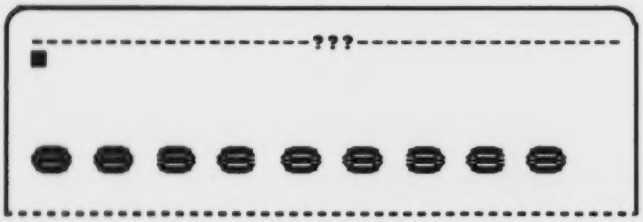
Your Shapes grid may look different. Each version of *LogoWriter* has a grid of a different size.

When you have created your shape, experiment to see how many of your new shapes fit across the page. Use Turtle-move to place the turtle. Experiment with different distances between the shapes by modifying the above **repeat** instruction.

```
repeat 11 [pd stamp pu forward 25]
```



```
repeat 9 [pd stamp pu forward 29]
```



Decide how many shapes will fit and still leave room for your labels.

Next you need to determine what number each symbol represents. Look at the examples at the beginning of this chapter. One basketball symbol represents 3 points scored. One hamburger symbol represents 1 million hamburgers sold.

Choose the largest number from your data chart. Use it to find out how much each symbol is worth. If the row of pictures representing the largest number fits on the page, then all the rows will fit on the page.

Use *LogoWriter* to divide the largest number by the number of symbols that you want to fit across the page.

```
show 6239000/10
623900
```

Did you remember to leave space for your labels?

In most pictographs, each symbol represents a number that is easy to use in mental calculations. It is better to have a symbol represent 60 items than 58. It is better to have the symbol represent 1000 rather than 800. To avoid having too many symbols on the page, you should round your number up—make it larger.

In the hamburger example, one symbol represents 1 million. It was rounded up quite a bit. This allows larger numbers to be put on this graph later and leaves space for labels. The more you round up, the more symbols fit on the page.

Finally, check your calculation to be sure that the largest number from your data chart will fit on the page. First round off the largest number.

```
6,385,000  becomes 6 million
```

Six million means there will be 6 hamburger symbols. Six symbols will fit on the page. There will also be space for labels.

Make sure *your* rounded off number is less than the number of symbols you want on *your* page. Figure out how many symbols you will need for each number in your chart. Add a column to your chart.

Number of hamburgers sold per week		Number of symbols
Wong's	6,239,000	6
TacoMex's	2,845,000	3
Angelo's	5,335,000	5
O'Connery's	2,007,000	2
Pierre's	2,700,000	3

Now you are ready to create your pictograph.

Draw the Rows of Pictures

You can use **repeat** in the Command Center to put the rows of pictures on the page. Or, you can write a procedure to put the rows of pictures on the page. If you write a procedure, you should use an input so that one procedure will work for any number of pictures.

```
to picture.bar :no.of.pics
repeat :no.of.pics [pd stamp pu forward 25]
end
```

Now you can type

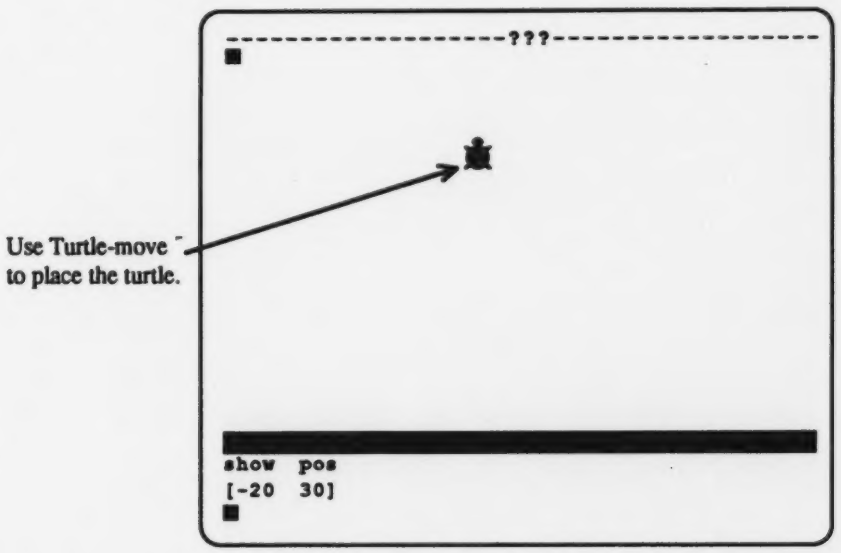
```
picture.bar 5
```

to create a bar that has 5 pictures.

You can use Turtle-move to place each bar of the graph. Be sure to leave room for labels. You can use

```
show pos
```

to get the coordinates for the beginning of a bar.

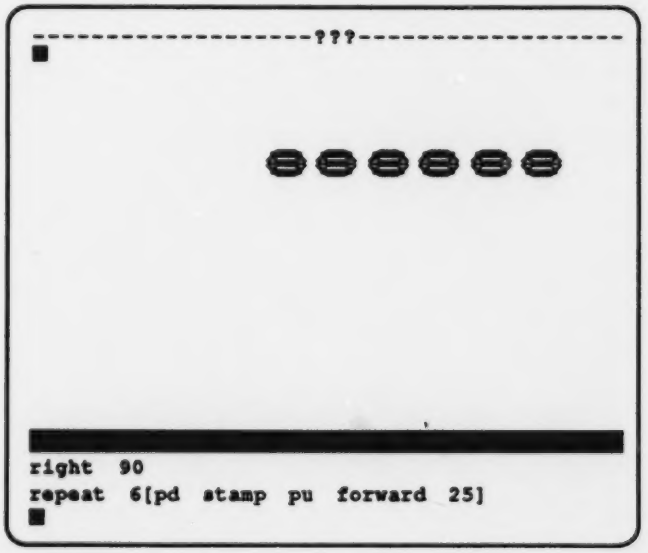


Then, use the coordinates to place the turtle before drawing each row.

```
setpos [-20 40]
right 90
repeat 6 [pd stamp pu forward 25]
```

or

```
setpos [-20 40]
right 90
picture.bar 6
```



Did you get an error? Be there is no space in **setpos**? Did you use square brackets after **setpos**?

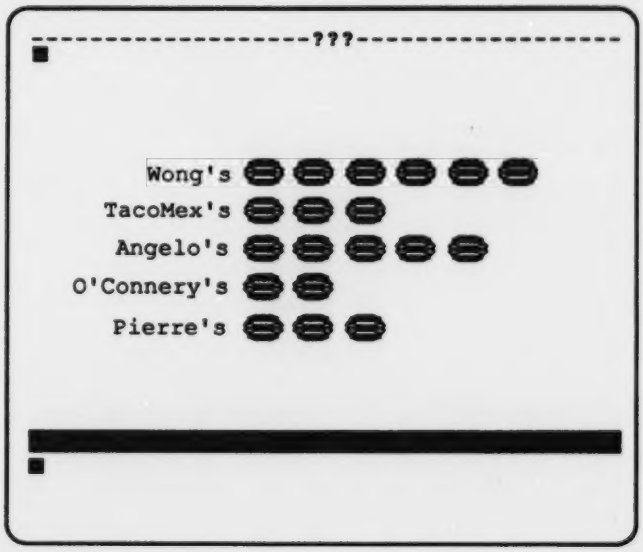
Does the row go up and down? Be sure to turn **right 90** before you draw the bar.

Now draw all of the bars for your graph.

Add Labels to Your Graph

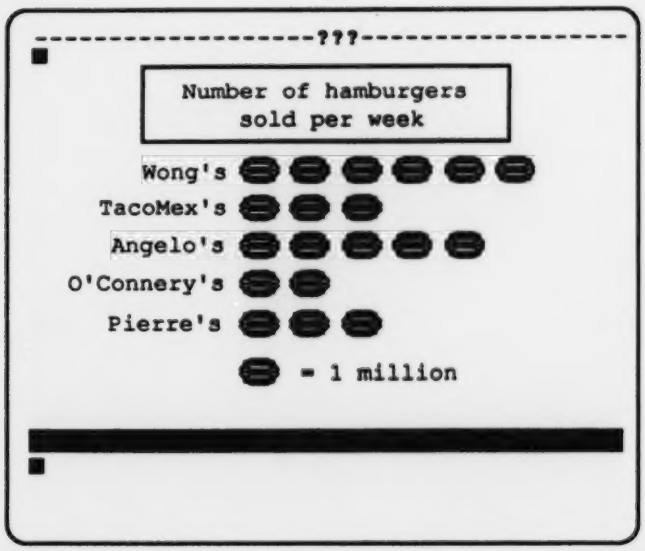
Next you need to add labels to each of the rows of your graph. This is a good place to use **label**. You can either use the Label keys or the **label** command. Labels are graphics characters rather than the kind of text produced by **print** or the word processor. Label text becomes part of your drawing. Remember that

- the Label keys put text at the position of the label cursor, and that
- the **label** command puts text at the position of the turtle. If you use the **label** command, find the position of each label using Turtle-move.



Finish your graph

When you finish the main part of your graph, you can add a title and a "key" that shows the value of each symbol. You probably want to use **label** for the text of the title and the key. Don't forget to hide the turtle when you have your graph completed.



There are a number of things you can do to make your graph more interesting. You may want to add a box around the title or the key. How about making a border around the entire graph? You might even want to add some animation that begins when you get the page. What other extras can you think of to make your graph more attractive, fun, and easy to read?

When you are finished, be sure that you have saved your page. Perhaps you also want to print your graph.

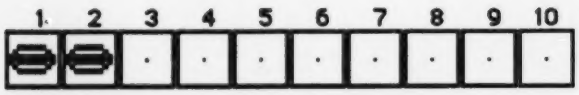
Making Half-shapes

Pictographs in newspapers and magazines often have partial symbols. For example, in the above graph, one-half of a hamburger symbol would represent one-half million hamburgers. You can easily create half-shapes to use in your graph.

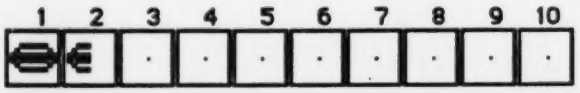
Suppose the shape for your graph is in Shape 1.

1. Go to the Shapes page.
2. Go to Shape 1.
3. Copy Shape 1.

4. Go to Shape 2—or some blank shape on the Shapes page.
5. Paste your graph shape into Shape 2.



- Erase half of your shape.



You now have a half-shape.

> Your Shapes page may look different. The Shapes page in each version of *LogoWriter* is slightly different. You may have used different shape numbers for your shapes and half shapes.

Next, you must scale your half-shape. Multiply the value of one shape times one-half. For example:

```
show 1000000 * 0.5
500000
```

If one shape represents a million, then a half-shape represents 500,000.

Now compute how many half-shapes you need. The whole number is the number of complete shapes.

```
show 6290000/1000000
6.29
```

Compute the amount left over. First find out how many your row of shapes represents.

```
show 6 * 1000000
6000000
```

Subtract the amount represented by your row of shapes from the original data.

```
show 6290000 - 6000000
290000
```

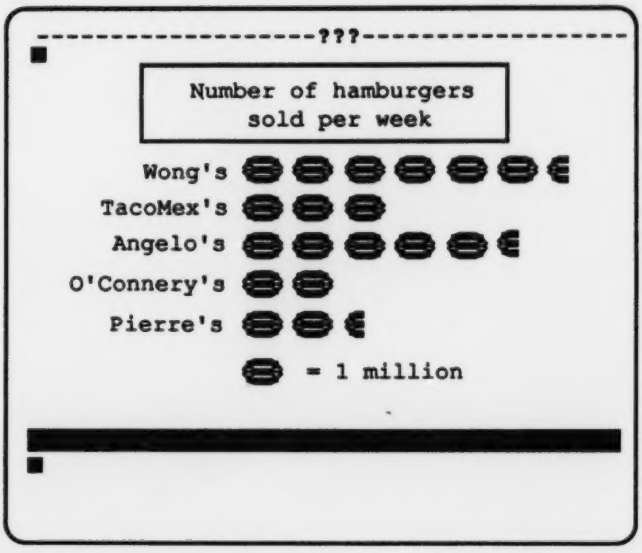
Now divide what is left by the value of your half-shape.

```
show 290000/500000
0.58
```

If the answer is bigger than one half, then add a half-shape.

```
show 0.58 > 0.5
true
```

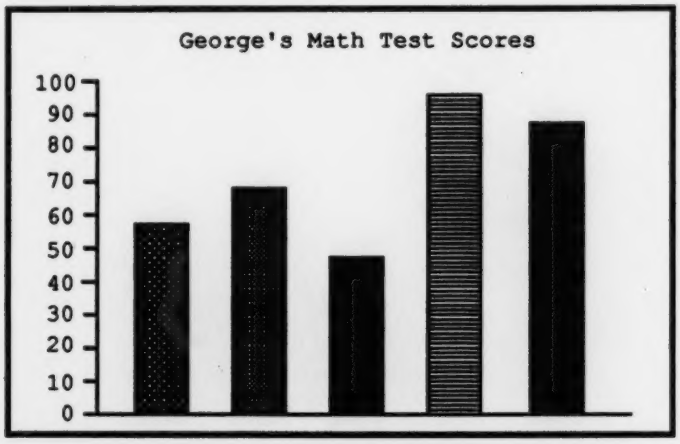
Using partial shapes, the hamburger graph from above looks like this.



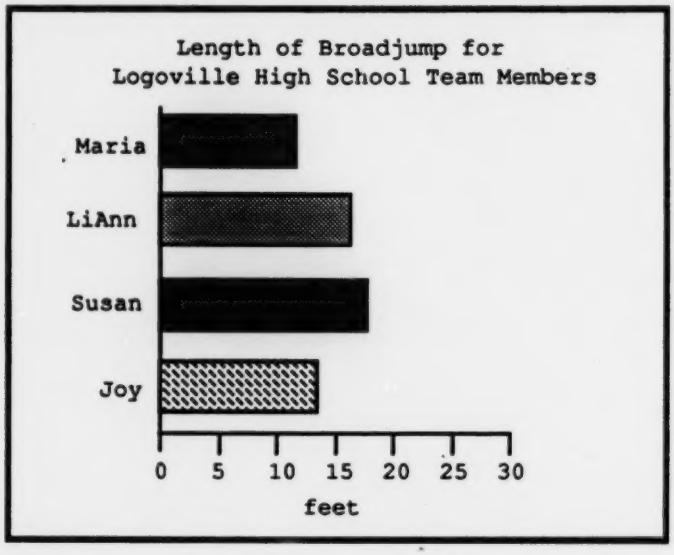
You can also make quarter shapes. How would you find the value of a one-quarter shape? How would you figure out how many quarter shapes you need? Partial shapes make a pictograph more accurate.. But partial shapes can make a pictograph harder to read. Be careful. Don't use too many partial shapes.

3. Make a Bar Graph

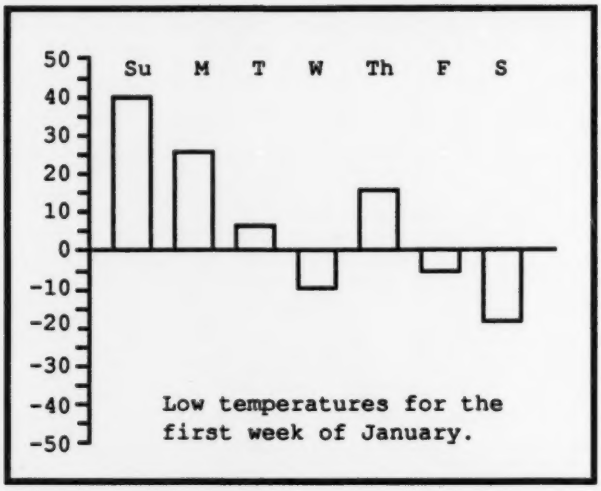
Bar graphs are used in many places. You have probably seen them in your textbooks at school. You have seen them in newspapers and magazines.



A bar graph represents data more accurately than a pictograph.



You can make a bar exactly the right height. With a pictograph you usually round numbers so that you only have whole picture-symbols or perhaps half picture-symbols.



Start Your Bar Graph

Making a bar graph is similar to making a pictograph. Many of the steps are the same. The first step is to collect data for the bar graph. You should put the data into a chart so that it will be easy to use when you get ready to make your graph.

George's Test Scores
58
68
49
96
87

Next, you need to make a bar. You can use the **repeat** command.

```
repeat 2 [forward 40 right 90 forward 10 right 90]
```

You need to write a different **repeat** instruction for every bar.

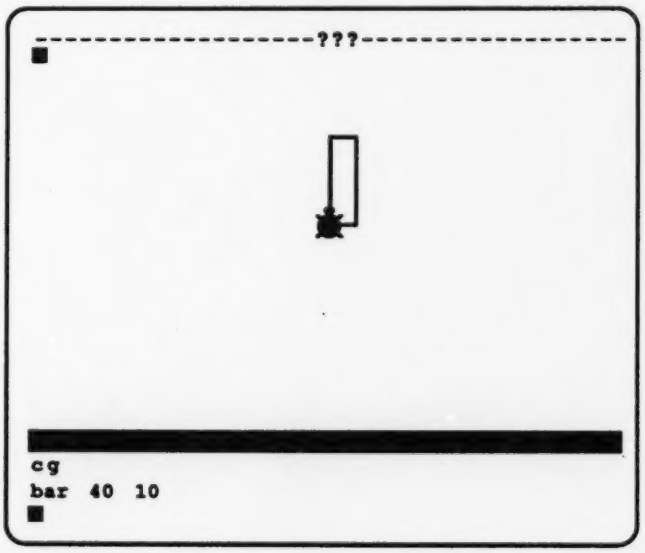
You can save time by writing a procedure to make a bar. You should be able to enter the length (or height) and the width of the bar. This means you need a procedure with two inputs.

```
to bar :length :width
repeat 2 [forward :length right 90 forward :width
  right 90]
end
```

Typing

```
cg
bar 40 10
```

creates a bar like this.



Scale Your Bars

Next, you need to decide how to make bars for your data that will fit on the page. This process is similar to scaling pictographs. But bars in a bar graph can be more exact. You do not have to worry about how many picture-symbols fit across the page.

First, find the largest number in your data list. If the bar representing the largest number will fit on the page, then all the other bars will fit on the page.

Will your bars go up and down? Or will your bars go across. Decide on the direction of your bars. Then you can decide how many turtle steps the largest bar can be. Remember that you will need space for any labels or titles that you put on the graph.

Divide the biggest bar height by the largest item in your data list.

```
show 150/96
1.5625
```

Round this number. This tells you the number of turtle steps needed to represent one unit on your graph. Be sure the largest number in your data list will create a bar that fits on the page.

```
show 1.6 * 96
153.6
```

Adjust the amount representing each unit on your graph, if necessary.

```
show 1.5 * 96
144
```

Next, extend your data chart. Make a column for the number of turtle steps. Multiply each data item by the number of turtle steps that represents one unit.

```
show 1.5 * 58
87
```

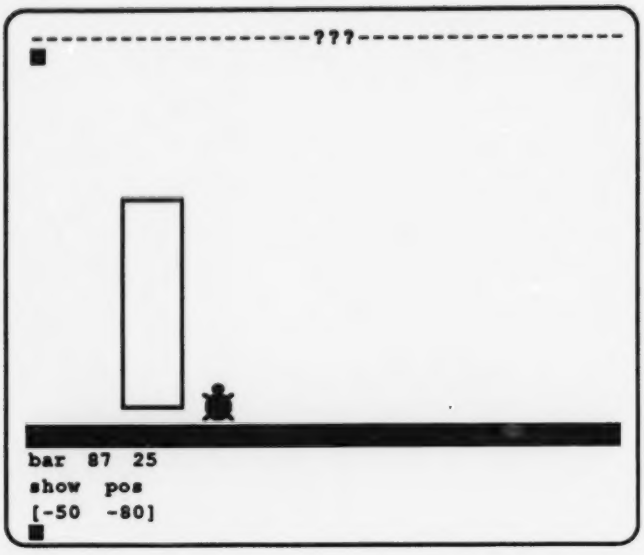
Round these numbers if needed.

George's test scores	Number of turtle steps
58	87
68	102
49	74
96	144
87	131

Place Your Bars

Now you can draw the bars in the bar graph. Choose the position of the first bar. Use **setpos** to place the turtle to draw the first bar. Here are two ways to place the rest of the bars.

- Use Turtle-move and **show pos** to locate each bar. Then use **setpos** to place the bar.



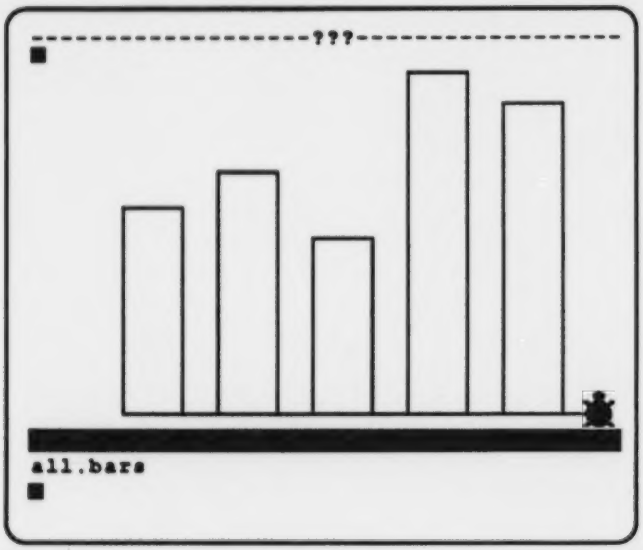

```
cg
pu
setpos [-90 -80]
pd
bar 87 25
pu
setpos [-50 -80]
pd
bar 102 25
```
and so forth.

- Use turtle graphics commands such as **right** and **forward** to place each bar. Write a procedure to move to the next bar.

```
to next.bar
right 90
forward 40
left 90
end
```

```
cg
pu
setpos [-90 -80]
pd
bar 87 25
next.bar
bar 102 25
next.bar
```
and so forth.

The **next.bar** procedure keeps the pen down so that the turtle draws a base line for the graph. Can you plan your procedure so that a base line is drawn?



You may need to experiment with spacing of your bars. You can

- change the width of each bar, or
- change the distance between bars.

Is the graph too wide? Make the width of each bar or the distance between the bars smaller. Is the graph too narrow? Make the width or distance larger. When you are satisfied with your bar graph, be sure to save your page.

Using procedures makes it easier to change your work. You can write an **all.bars** procedure using **setpos**

```
to all.bars
cg
pu
setpos [-90 -80]
pd
bar 87 25
pu
setpos [-50 -80]
pd
bar 102 25
 and so forth.
end
```

or using **forward** and **right**

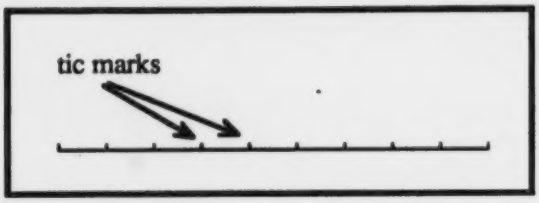
```
to all.bars
cg
pu
setpos [-90 -80]
pd
bar 87 25
next.bar
bar 102 25
next.bar
 and so forth.
end

to next.bar
right 90
forward 30
left 90
end
```

If you write a procedure to draw all the bars, test it to be sure that it works as you planned. Be sure the bars are spaced the way you want.

> Remember, you can adjust the width of the bars or the distance between the bars. You have scaled the bar height. You can't change it. Your graph will not be accurate.

Make an Axis

Most bar graphs have an axis with a scale to help show what numbers the bars represent. Often axes are shown with "tic marks" to indicate the scale of the graph.

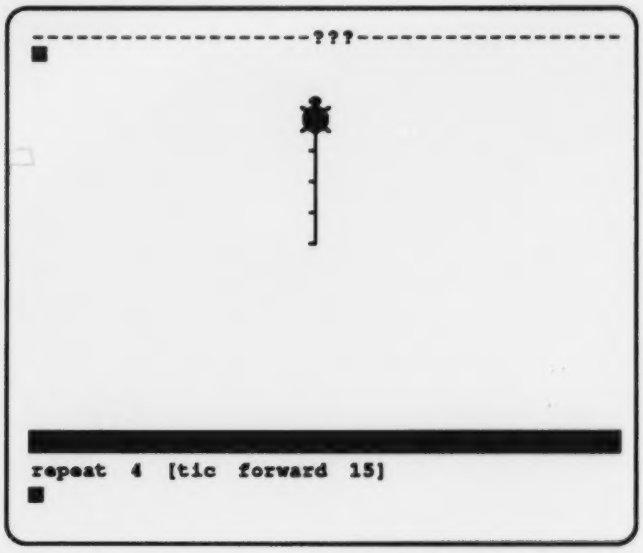


You can write a **tic** procedure.

```
to tic
left 90
forward 2
back 2
right 90
end
```

Use a **repeat** instruction that includes **forward** and **tic** to create an axis.

```
repeat 4 [tic forward 15]
```

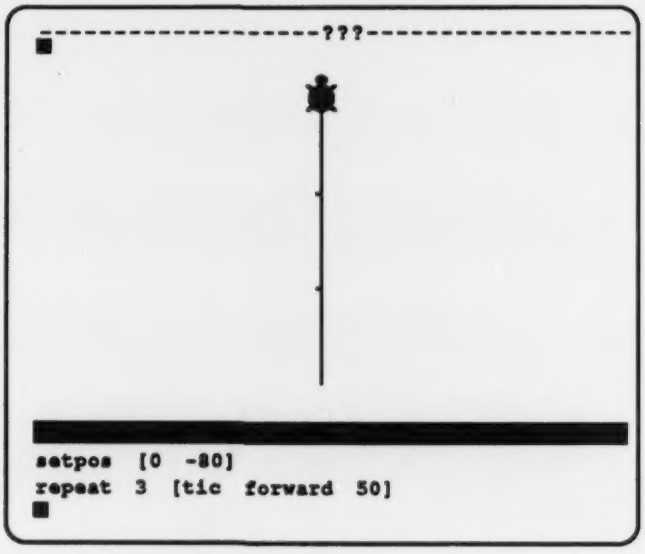


Scale Your Axes

You need to scale your axes to match your bars. You have chosen the maximum size for your bars. You computed the number of turtle steps per unit. Now, you need to decide the distance between tic marks. Usually a number like 5, 10, or 100 is used for these spaces.

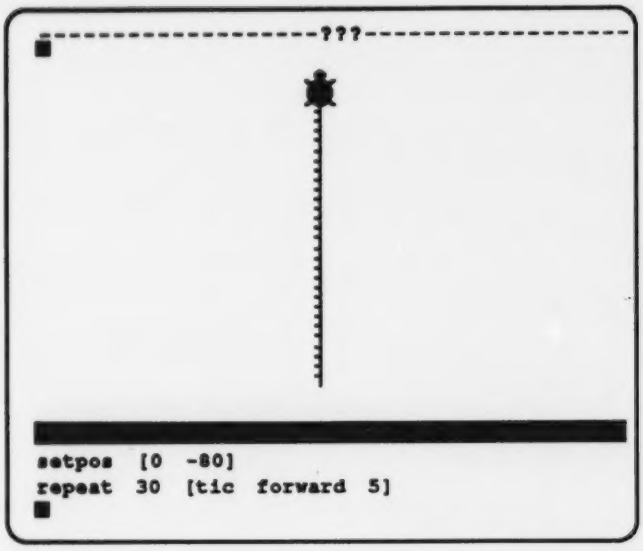
Think about the way your scale will look. Suppose your longest bar is 150 turtle steps. Then, using 50 turtle steps between tic marks means that there will be only a few tic marks on the axis.

```
show 150/50
3
```



If you use 5 turtle steps between tic marks, there will be *many* tic marks.

```
show 150/5
30
```



The space between tic marks should be a number that is easy to use in mental arithmetic. Use numbers like 5, 10, or 100. Select a number to use for your graph. Use your largest data value to find the number of tic marks.

```
show 96/5
19.2   means about 20 tic marks
```

Remember that you computed the number of turtle steps per unit.

```
show 150/96
1.5625    became     1.5
```

Find the distance between tic marks in turtle steps. Multiply the space between tic marks by the turtle steps per unit.

```
show 5 * 1.5
7.5
```

Are you happy with the space between tic marks. Tic marks that are too close together make a graph hard to read. Tic marks that are too far apart are not of much use.

If necessary, adjust the number of tic marks you are going to use.

```
show 10 * 1.5
15  turtle steps between tic marks
```

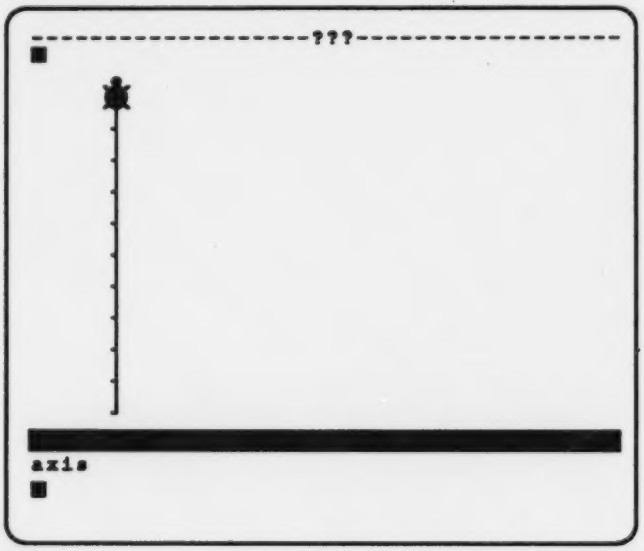
Then draw an axis.

```
setpos [0 -80]
repeat 10 [tic forward 15]
```

> The number after **repeat** is the number of tic marks you are using. The number after **forward** is the number of turtle steps between tic marks.

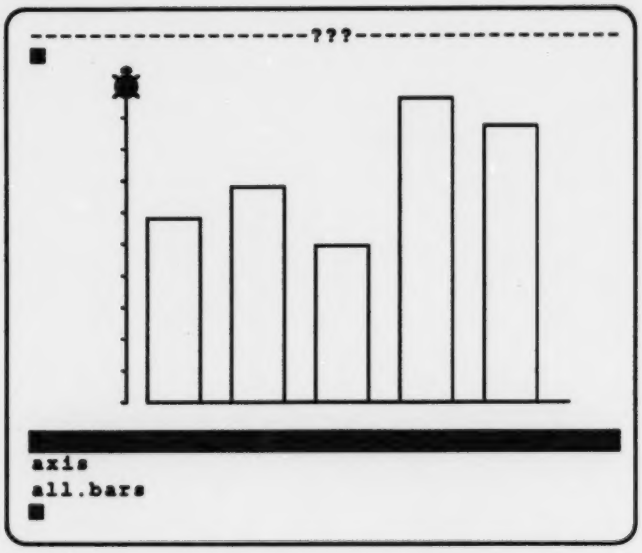
Now you are ready to write an axis procedure.

```
to axis
pu
setpos [-100 -80]
pd
repeat 10 [tic forward 15]
end
```



> Does your axis go off the screen? Check your computations.
>
> Are your tic marks too close together? Are your tic marks too far apart? Try a smaller or larger distance between tic marks.
>
> Remember, you must keep the turtle steps per unit the *same* so the bars matches the axis.

Did you write an **all.bars** procedure? Add this **axis** procedure.



You may want to write a procedure for your entire graph. Add a **set.up** procedure to prepare the page for drawing.

```
to bar.graph
set.up
all.bars
axis
end

to set.up
ct
cg
ht
end
```

Label Your Axes

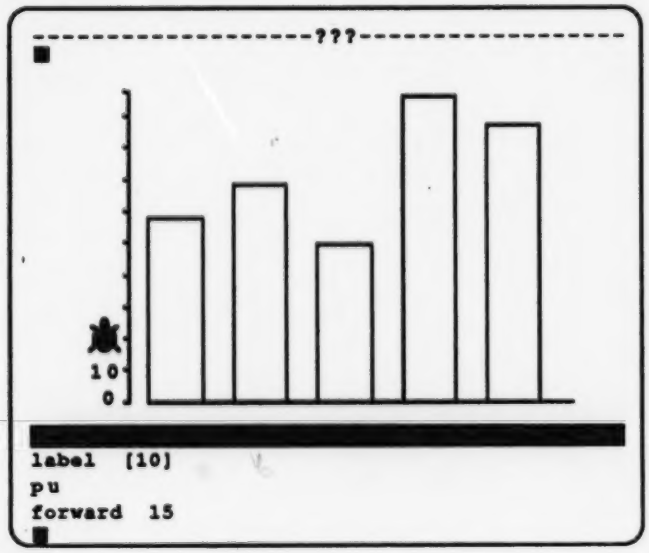
Some bar graphs have a "key" like pictographs. Often bar graphs have their axis labeled.

- You can use **setpos** to place the turtle and **label** to put the numbers on the scale.

```
pu
setpos [-120 -85]
pd
label [0]
pu
setpos [-120 -70 ]
pd
label [10]
pu
```
and so forth.

- You can use **forward, back, right,** and **left** to move the turtle from tic mark to tic mark and then use **label** to add the numbers.

```
pu
setpos [-120 -85]
pd
label [0]
pu
forward 15
pd
label [10]
pu
forward 15
```
and so forth.



You can write a procedure to make the scale. You can add it to your **all.bars** procedure.

Color Your Bars

Often the bars in bar graphs are shaded or filled with color. You can shade or fill your bars in several ways.

- You can make colored bars. Move the turtle with Turtle-move so that the *center* of the turtle is over the open part of the bar. Set the pen color with **setc** and type:

```
pd
fill
pu
```

or pick a shape that will make an interesting pattern and type:

```
pd
shade
pu
```

- You can add an input to your **bar** procedure that allows you to select a color (or shape) each time you call the procedure.

```
to bar :length :width :the.color
setc :the.color
repeat 2 [forward :length right 90 forward :width
  right 90]
end
```

Then you can move the turtle and fill the bar.

- You can write a procedure to fill a bar

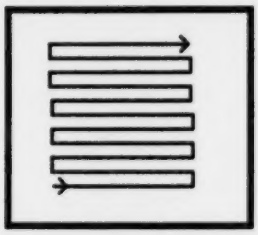
```
to fill.bar  :bar.width
pu
right 45
forward :bar.width/2
pd
fill
back :bar.width/2
left 45
end
```

and add it to your **bar** procedure.

```
to bar :length :width :the.color
setc :the.color
repeat 2 [forward :length right 90 forward :width
  right 90]
fill.bar :width
end
```

*Self-filling Bars

Filling bars with color or patterns can be difficult if the bars are very narrow. Another way to draw a bar is to have the turtle draw the bar and color it at the same time. That is, you direct the turtle to go back and forth so that it fills the bar. The path the turtle takes is something like this:



Think for a moment about writing a procedure to accomplish this task.

- The turtle moves to the right.
- Then the turtle turns left.
- Then the turtle goes "up" a small amount.
- Then the turtle turns left.
- Then the turtle moves to the left.
- Then the turtle moves "up" again.

Can you write this procedure?

Here is one solution:

```
to filled.bar :height :width
right 90
repeat :height [forward :width left 90 forward 1 left 90
  forward :width right 90 forward 1 right 90]
end
```

In the **repeat** instruction, the turtle moves "up" 2 turtle steps. Thus typing

```
bar 20
```

produces a bar that is 40 turtle steps high. One way to use this procedure is to scale your data differently. First multiply by the scale factor.

```
show 96 * 1.5
144
```

Now divide the new number by 2.

```
print 144/2
72
```

Add another column to the data chart that you made earlier.

George's test scores	Number of turtle steps	Input to bar
58	87.0	43.50
68	102.0	51.00
49	73.5	36.75
96	144.0	72.00
87	130.5	62.25

You will probably want to round the numbers in the final column. Now modify your **all.bars** procedure accordingly.

There are other ways to change the bar.graph program when using the **filled.bar** procedure. Can you think of a different method than the one described above?

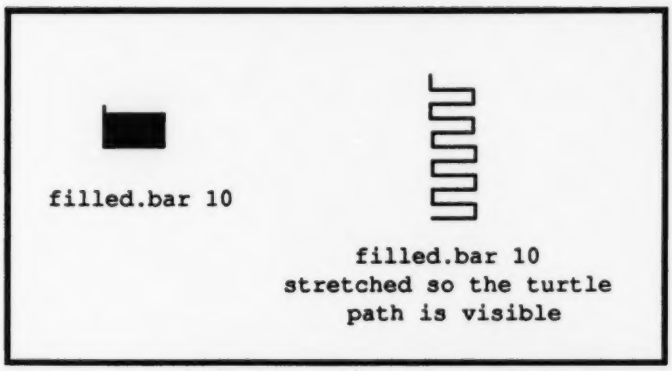
*Debugging Filled Bars

There are a number of things that can go wrong when drawing bar graphs using this procedure. Here are some problems and some possible solutions.

If you use the self-filling bar and type,

```
filled.bar 10
```

you see



The **filled.bar** procedure ends with the turtle going forward 1 step.

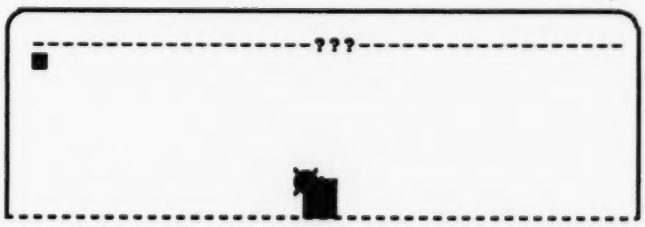
- You need to modify the procedure. You can use **px** to erase the last bit of line.
- You can reduce the number after **repeat**. Then the last part of **filled.bar** becomes

```
forward :width
left 90
forward 1
left 90
forward :width
```

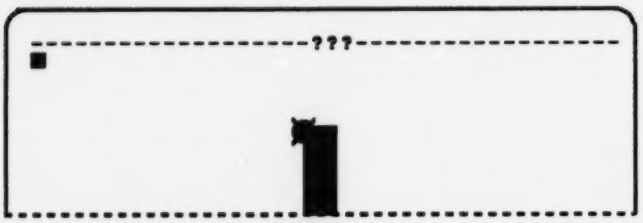
Another problem occurs when you try to replace the old **bar** procedure with the new self-filling one. If you type,

```
filled.bar 30
filled.bar 50
```

you see



and then



You need to get the turtle back to the place where it started at the end of the **filled.bar** procedure. This is called *state transparency*. It is always a good idea to make your procedures state transparent. Return the turtle to the place where it started. One way to do this in **filled.bar** is to add

```
left 90
back :height * 2
```

Finish your Graph

Finally, you need to add text to the graph. Your graph should have a title. It should probably have text identifying each bar. Use **label**. It creates graphics that will "stick to" the graph. The label text will not move around if the word processor is used to add information to the graph page.

Do you have a large number of labels to add to your graph? You can write a general labeling procedure that allows you to put labels anywhere on the page. The procedure below has two inputs.

```
to put.label :spot :text
pu
setpos :spot
pd
label :text
end
```

The first input is the location of the label. **Setpos** needs a list of two numbers as input. The input "spot" must be a list of two numbers. **Label** also needs a list as input. The "text" input must also be a list. For example, you can type:

```
put.label [-50 80] [George's Test Scores]
```

The words "George's Test Scores" will appear with the first letter at the point (-50, 20).

Finish your graph so that it is attractive and easy to read. Add color. Add patterns. Add a border. You might have a second page of text explaining the graph.

*Generalizing Your Bar Graph

You probably made your bar graph for a particular set of data. You might have similar sets of data that can make use of the program you wrote. This will work if you have data similar to the data for which you developed these procedures. If it is quite different, the bars may not fit on the page.

You can write a procedure that will accept a list of all of your data. It will graph one bar for each number in the list. Here is one way to write this procedure.

```
to graph :data.list :width
if empty? :data.list [stop]
bar first :data.list :width
right 90
forward :width + 10
left 90
graph butfirst :data.list :width
end
```

To use this procedure, you type:

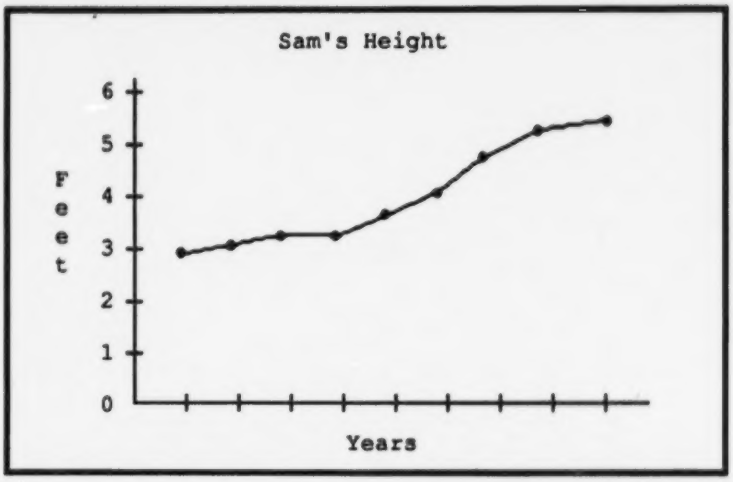
```
graph [44 51 37 72 65] 20
```

Five bars that are 20 turtle steps wide are drawn.

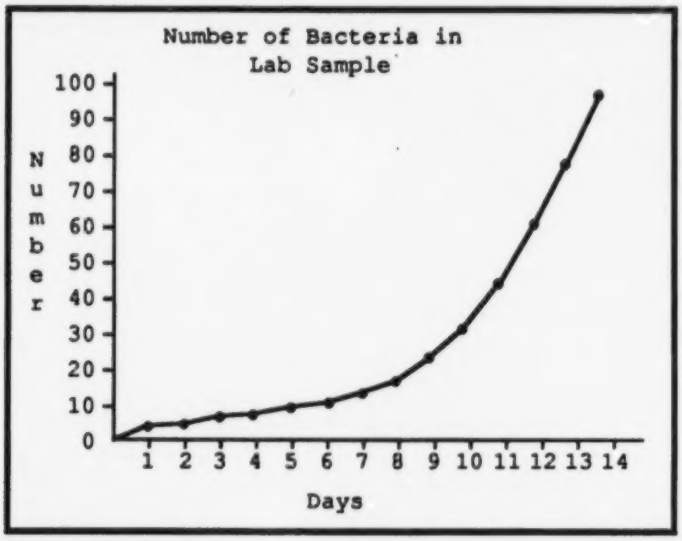
There are many ways that the graph procedure can be extended. Can you figure out how to add a list of colors as input to this procedure so that each bar is a different color. How about having bars of different widths?

4. Make a Line Graph

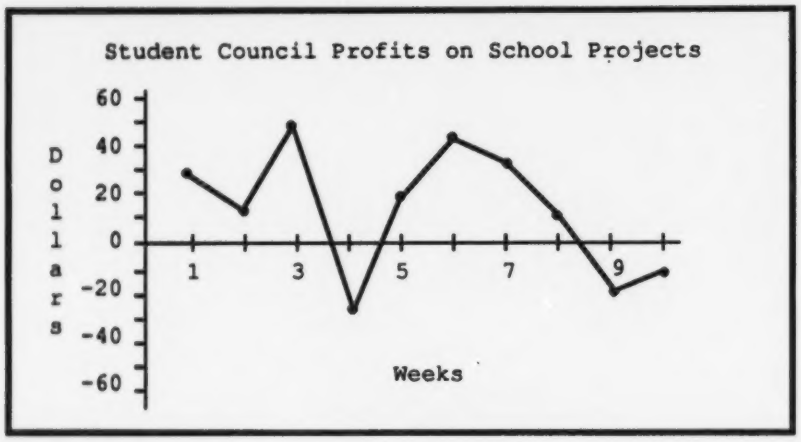
You have seen line graphs in a number of your school books.



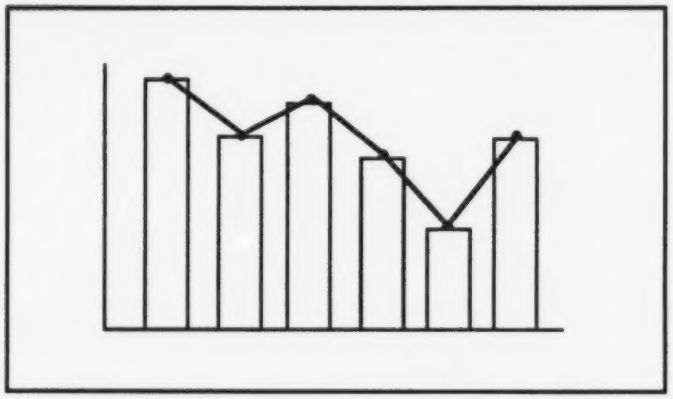
Line graphs are frequently used in math and science classes. These graphs usually have a scales on the bottom and the side of the graph.



Line graphs are also used to represent data in newspapers and magazines.



You can display some data on either line graphs or bar graphs. In this case, you can think of a line graph as connecting the tops of the bars in a bar graph.



Start Your Line Graph

The first step in creating a line graph is to collect some data. You can use data from a science experiment or an exercise in math class. Put your data into a chart for later use.

Day	Number of Bacteria
1	4
2	5
3	7
4	10
5	11
6	13
7	14
8	16
9	21
10	32
11	45
12	67
13	78
14	97

Create Axes

First you need axes for you graph. In most line graphs, you need both a horizontal (across) and vertical (up and down) axis.

Start with the horizontal axis. The first column in your data table usually goes on the horizontal axis. The second column goes on the vertical axis. Scaling your data for a line graph is similar to scaling data for a bar graph.

Determine how wide (in turtle steps) you want your graph to be. Divide the number of turtle steps by the largest number in your data table.

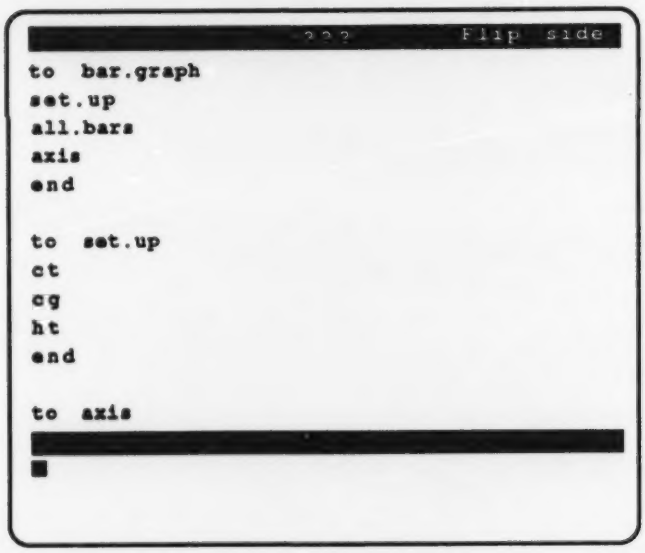
```
print 240/14
17.142
```

Round this number. This gives the number of turtle steps between tic marks.

Use a procedure to draw the axes for your line graph. You can load your **axis** procedure from your bar graph. The command **load** puts the procedures from another page onto your page. If you type something like

```
load "bar.graph
```

you see your bar graph program.



Use select and cut to remove any procedures you don't need. Does your **axis** procedure use **tic**? Be sure to keep **tic**.

The **axis** procedure was something like this.

```
to axis
pu
setpos [-100 -80]
pd
repeat 10 [tic forward 15]
end

to tic
left 90
forward 2
back 2
right 90
end
```

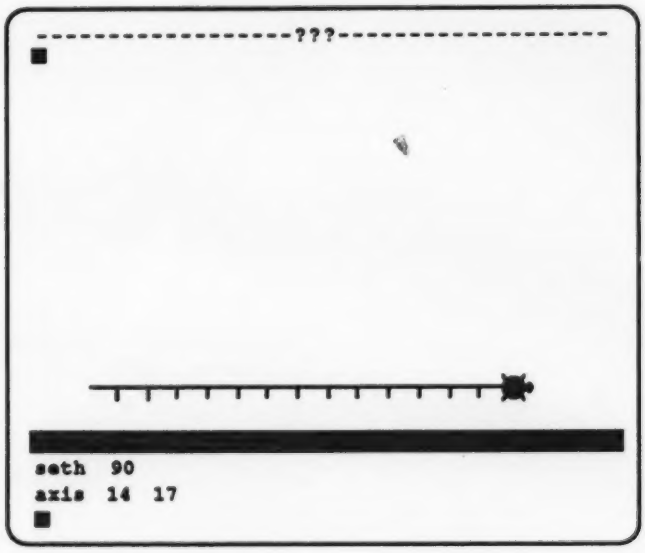
Modify the **axis** procedure so that it has one input for the number of tic marks and one input for the distance between tic marks.

```
to axis :number :distance
repeat :number [tic forward :distance]
end
```

Now place the turtle. Then

```
seth 90
axis 14 17
```

draws a horizontal axes.



Write a **horizontal.axis** procedure.

```
to horizontal.axis
pu
setpos [-120 -75]
pd
seth 90
axis 14 17
end
```

Scale the vertical axis in the same manner.

```
print 150/97
1.5464
```

Did you get a very small number of turtle steps between tic marks? Perhaps you want a tic mark every 10 or 50 or 100 units on your axis. Multiply to get an appropriate number of turtle steps.

```
show 10 * 1.5
15  turtle steps between tic marks
```

Make sure that your largest data value will fit on the page. Divide the largest data value by the number of units you choose.

```
show 97/10
9.7 tic marks
```

Round. Multiply by the number of turtle steps.

```
show 10 * 15
150
```

> Will your axis fit on the page? If not, make the number of turtle steps between tic marks smaller.

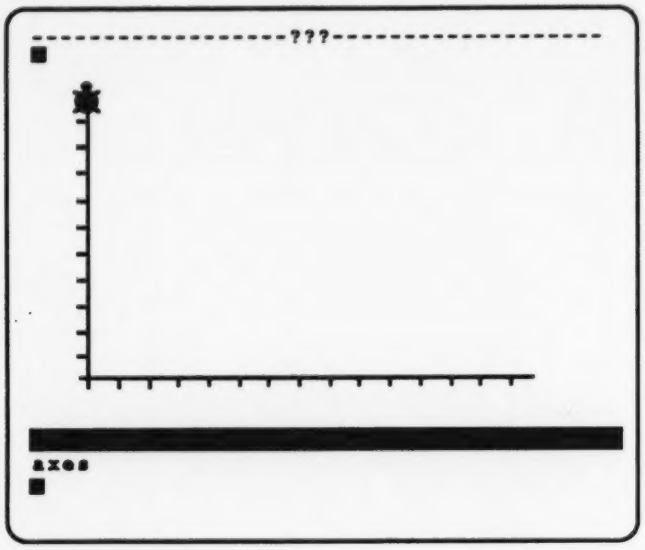
Now write a procedure to draw your vertical axis.

```
to vertical.axis
pu
setpos [-120 -75]
pd
seth 0
axis 10 15
end
```

Make sure you have the right number of tic marks. Make sure your axis fits on the page.

When you are satisfied with both axes, write a procedure to draw both axes at the correct location on the page.

```
to axes
vertical.axis
horizontal.axis
end
```



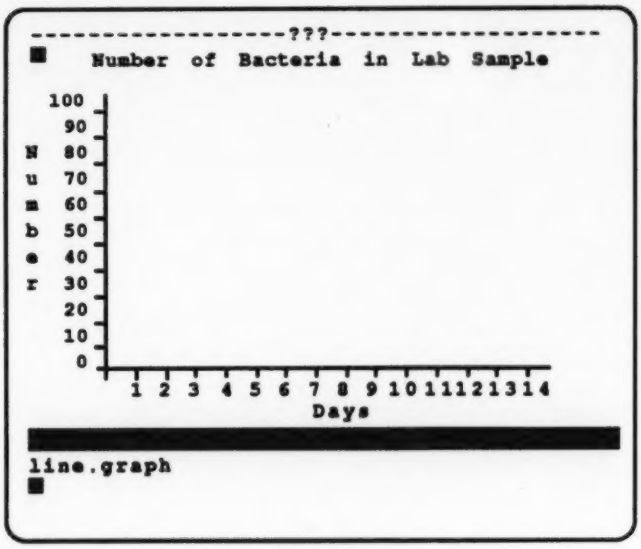
Label Your Graph

There are three things you should add when you label your graph.

1. Identify how much each the space between tic marks represents. Put numbers along the vertical and the horizontal axes.
2. Put labels on each axis. Are you using feet or meters? Are you using days or hours? Or are you using some other units?
3. Put a title on your graph.

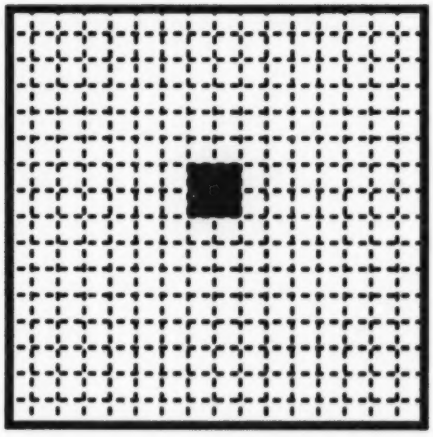
There are several ways you can put labels on your graph.

- You can use **label**.
- You can use the Label keys.
- You can use the **put.label** procedure from the Bar Graph chapter.



Plot Your Data Points

First, you need a procedure to make a dot. Go to the Shapes page. Make a turtle shape that is a small dot. Usually four dots in the center of the turtle grid works fine.



Give the turtle your new shape.

```
setsh 1
```

48

Next you need a procedure to put the dot on the graph.

```
to make.point
pd
stamp
pu
end
```

Next, you need to find out where to put the dots. There are several ways to do this. You can have the turtle march along the horizontal axis. When it reaches a tick mark, it moves vertically to the correct spot.

To get from tic mark to tic mark on your graph, start where the axes cross.

```
pu
setpos [-120 -75]
seth 90
```

Move the turtle along the horizontal axis one tic mark.

```
forward 17
```

Turn the turtle so it will move parallel to the vertical axis.

```
seth 0
```

Move the turtle. Use the first number in your data list. Multiply by the number of turtle steps per unit for the *vertical* axis.

```
forward 4 * 1.5
```

The turtle is now at the first point in your graph. Find its location.

```
show pos
[-103 -69]
```

Turn the turtle so that it is parallel to the horizontal axis.

```
seth 90
```

Move the turtle the distance of one tic mark.

```
forward 17
```

Since the turtle is already above the horizontal axis, you need only have it move the extra distance to the next data point.

```
seth 0
forward (5 - 4) * 1.5
```

Again find its position using **pos.**

Make a table to help find the points on the graph.

Day	Number of Bacteria	Turtle Steps	Position of the Turtle
1	4	4 * 1.5 = 6.0	[-103 -69]
2	5	(5 - 4) * 1.5 = 1.5	[-86 -67.5]
3	7	(7 - 5) * 1.5 = 3.0	[-69 -64.5]
4	10	(10 - 7) * 1.5 = 4.5	[-52 -60]
5	11	(11 - 10) * 1.5 = 1.5	[-35 -58.5]
6	13	(13 - 11) * 1.5 = 3.0	[-18 -55.5
			...and so forth

Now you can write a procedure to put the dots on the graph.

```
to line.graph
pu
setpos [-120 -75]
pd
make.point
setpos [-103 -69]
make.point
setpos [-86 -67.5]
make.point
```

and so forth.

A line is drawn between the points when the pen is down. Remove **pu** from the **make.point** procedure when you want the line drawn.

Writing procedure to draw the point saves time and typing. It also makes your program more readable.

*Another Way to Plot Points—Use Your Bar Procedure

Earlier in this chapter you saw that a line graph is similar to a bar graph. Connect the tops of bars of a bar graph. You have a line graph.

Use **load** to get the procedure that you used to create the bars in your bar graph. Make the bar into a single vertical line. Have the program tell you the position of the turtle when it gets to the top of the bar. You can use

```
show pos
```

Change the scale in your bar graph procedure so that it matches your line graph. A list of coordinates appears in the Command Center when you run the program.

You can use a procedure similar to the **graph** procedure.

```
to plot.points :point.list
if empty? :point.list [stop]
seth towards first :point.list
forward distance first :point.list
pd
make.point
plot.points butfirst :point.list
end
```

To use this procedure, type:

```
pu
plot.points [[-103 -69] [-86 -67.5] [-69 -64.5]]
```

Plot.points contains two new primitives. **Towards** takes a point as input. It reports the heading of the turtle when it faces that point. Move the turtle away from the center of the page. Type:

```
seth towards [0 0]
```

The turtle turns so that it faces the center of the page.

> **Towards** is not a primitive in Version 1.0 or 1.1 of *LogoWriter*

Distance reports the distance from the position of the turtle to the point given following **distance.** Move the turtle away from the center of the page. Type

```
forward distance [0 0]
```

The turtle moves the distance from its current location to the center of the page.

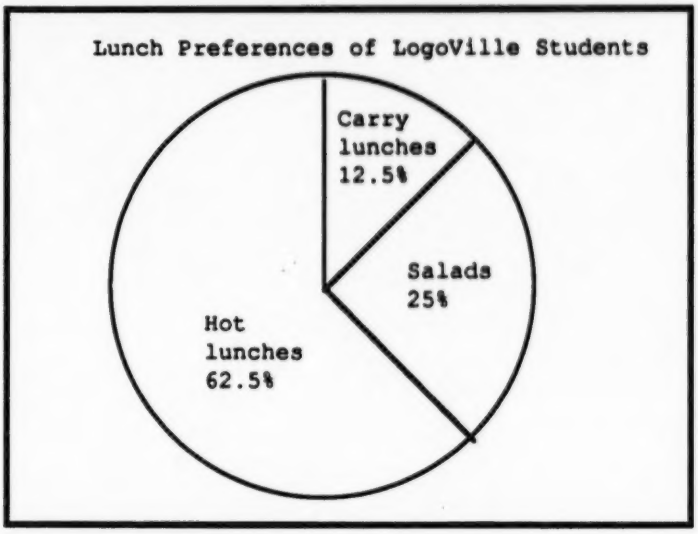
> **Distance** is not a primitive in *LogoWriter* versions 1.0 or 1.1. If you are using an Apple computer, you can find **distance** on the Mathtools page. You can load it onto your current page.

Finish Your Graph

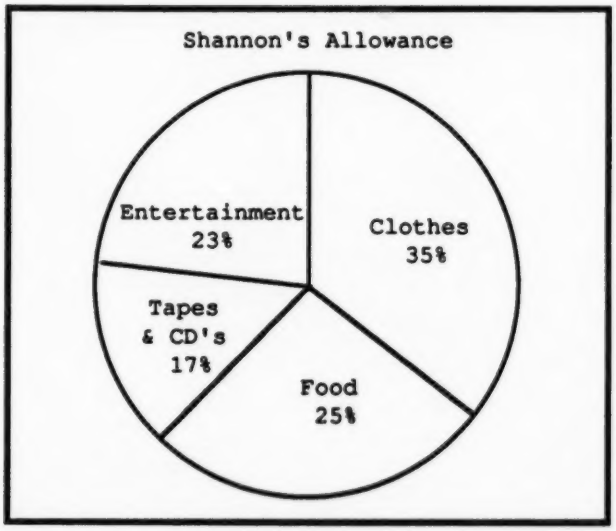
There are many additions that you can make to your graph. You can use color for lines or axes. You can plot points in one color and make the line another color. You can put a border around the graph. You can add a page of explanation about the graph. You might even want to have the graph drawn slowly as the user of your program watches.

5. Make a Circle Graph

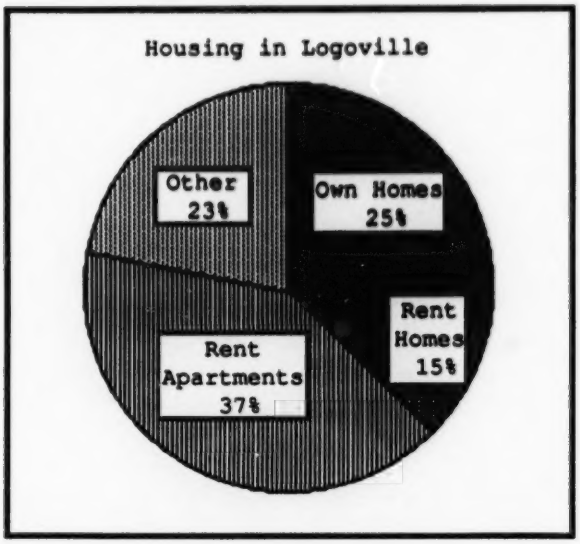
Circle graphs are used to show how the total is divided into parts. Each section of the graph shows a percentage of the area of the circle. You can use a circle graph to show the kinds of lunches eaten in your school cafeteria.



Or, you can use a circle graph to show how you spend your allowance.



You see circle graphs used in newspapers and magazines to show how each tax dollar is spent. You also see circle graphs used to show what percentage of the population live in particular kinds of housing.



Start Your Circle Graph

First, collect the data for your circle graph. There should be only a few categories of data. If there are too many separate pieces of data, the graph is hard to read.

Collect your data and put it into a chart.

Shannon's Allowance	
clothes	$5.25
food	$3.75
tapes & CD's	$2.55
entertainment	$3.45
	$15.00

Since circle graphs usually represent data using percents, add a percent column to your table. You can use *LogoWriter* to do the computations.

```
show (5.25/15.00) * 100
35
```

Shannon's Allowance		Percent
clothes	$5.25	35
food	$3.75	25
tapes $ CD's	$2.55	17
entertainment	$3.45	23
	$15.00	100

Check your percents to be sure that they add up to 100 percent.

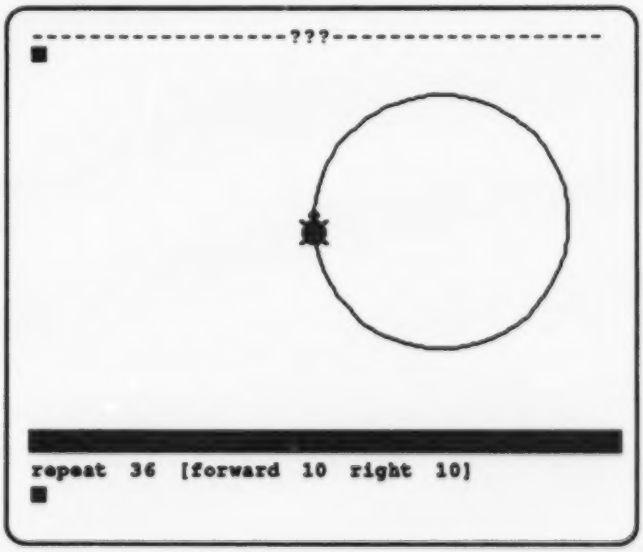
```
show 35 + 25 + 17 + 23
100
```

Draw a Circle

Have you drawn circles with Logo? Usually people draw circles using commands such as

```
repeat 36 [forward 3 right 10]
```

This is really a 36-sided polygon. But, it looks like a circle.



To draw a circle graph easily, you need to find the center. It is not easy to find the center of a circle drawn with the above instruction.

Here is another approach to drawing a circle. You can start the turtle at the center of the page,

```
cg
```

send it out a fixed distance,

```
forward 40
```

and draw a point.

```
pd
forward 0
pu
```

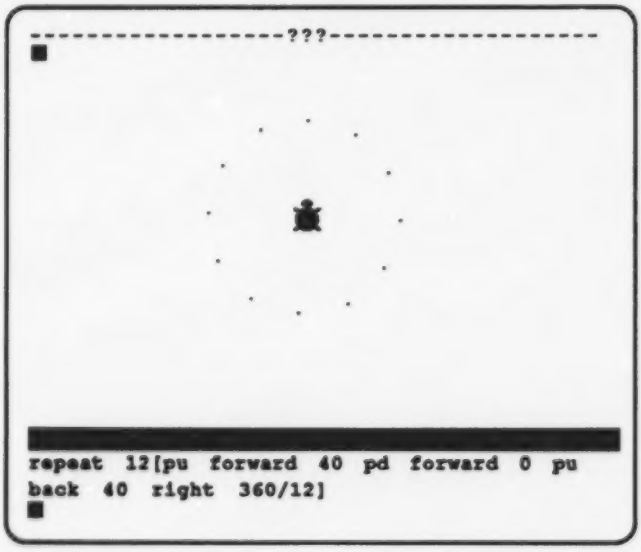
Move the turtle back to the center.

```
back 40
```

Turn the turtle a bit

```
right 30
```

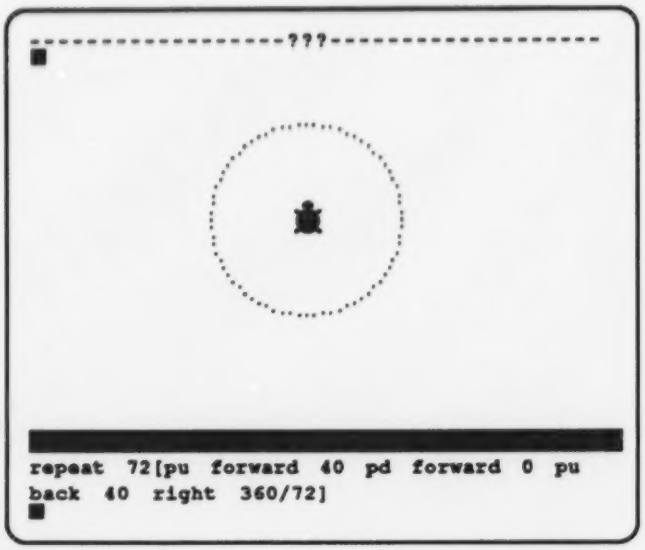
and repeat. Here is what happens if you draw 12 points.



Notice that moving the turtle forward zero turtle steps leaves a dot in *LogoWriter*.

To make the circle more solid, put the dots closer together.

```
repeat 72 [pu forward 40 pd forward 0 pu back 40 right
  360/72]
```



Experiment with this idea until you get a circle that looks solid.

Now you can write a circle procedure. You should use inputs. One input is the number of dots. The other input is the radius of the circle.

```
to circle :number.dots :radius
repeat :number.dots [pu forward :radius pd forward 0 pu
  back :radius right 360/:number.dots]
end
```

Typing

```
circle 72 40
```

draws a 72-dot circle with a 40 turtle-step radius.

Draw the Sections of the Graph

Now, you need to draw each of the sections of the graph. First write a **line** procedure. Use it to draw the dividers between parts of your graph.

```
to line :radius
forward :radius
back :radius
end
```

> Nothing happen? Did you forget to put the pen down (**pd**) before you typed **line**?

Typing

```
line 40
```

draws a line and returns the turtle to its starting point.

You need to compute the number of degrees needed in each section. Add another column to your data table.

```
show (35/100) * 360
126
```

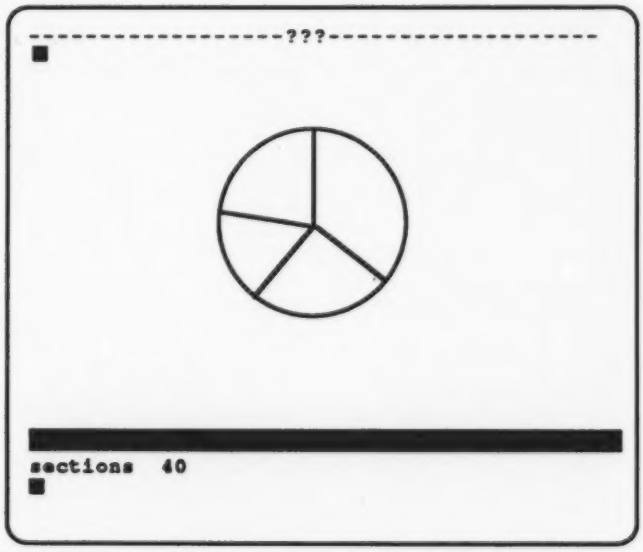
Shannon's Allowance		Percent	Degrees
clothes	$5.25	35	126
food	$3.75	25	90
tapes $ CD's	$2.55	17	61
entertainment	$3.45	23	83
	$15.00	100	360

Then you can write a procedure to draw the sections of the graph.

```
to sections :radius
line :radius
right 126
line :radius
right 90
line :radius
right 61
line :radius
right 83
end
```

Now you can write a procedure to draw your graph.

```
to circle.graph
circle 72 40
sections 40
end
```



Working With Color

Do you want to color the sections of your graph? Be sure to add color or patterns before you add your labels. Save a copy of your graph page before you experiment with color.

You can write a tool procedure for coloring sections of your graph.

```
to fill.area :place :color
pu
setpos :place
setc :color
pd
fill
end
```

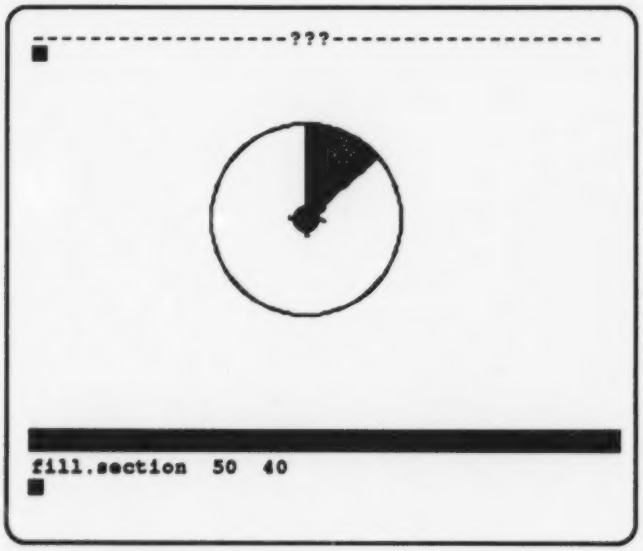
Then you can type

```
fill.area [20 10] 3
```

Sometimes this does not work. Are there are spaces between the dots in your circle? The color will "leak out" between those holes.

There is another way to fill your circle.

```
to fill.section :amount :radius
repeat :amount [forward :radius back :radius right 1]
end
```



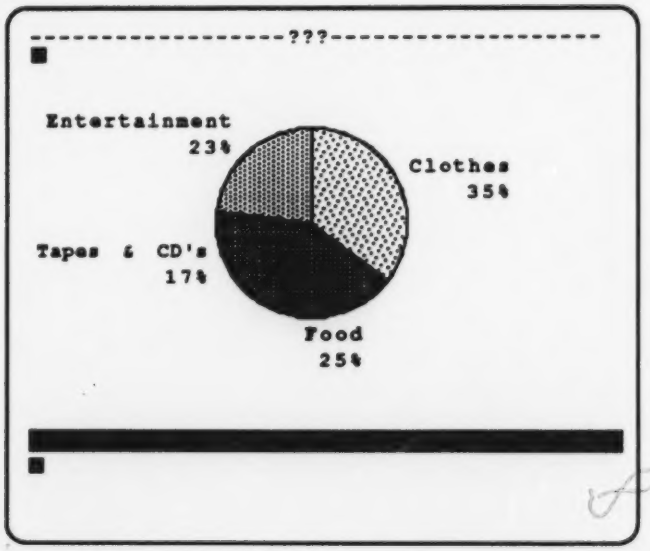
Note that you can also write **fill.section** like this:

```
to fill.section :amount :radius
repeat :amount [line :radius right 1]
end
```

Finish Your Graph

You now need to add a title and labels to your graph. You should use **label** for this. You can use the **place.label** procedure from the bar graph chapter.

In some versions of *LogoWriter* labels on colored areas are hard to read. Put your labels outside the graph.



What other features could you add to your graph? Here are some suggestions

- Fill the sections one at a time so that the person using the program can read each label and look at each part as it appears.
- Write a more general program into which you can put any set of data. Can you have it change the numbers into percents and then draw the graph?
- Combine a circle graph with one of your other types of graphs. You can save your data using

  ```
  make "data list [ your data list here ]
  ```

  You can then call a general circle graph with

  ```
  circle.graph :data.list
  ```

  Then, you can get another page, perhaps your bar graph page and type

  ```
  bar.graph :data.list
  ```

6. Which Graph Should I Use?

In this booklet, you have seen four kinds of graphs.

- pictographs
- bar graphs
- line graphs
- circle graphs

When do you use each kind of graph? Some data will look fine on any of the graphs. Some data will look fine on several kinds of graphs. There are some "rules of thumb" that you should consider.

Pictographs

Pictographs can be read quickly. They are interesting to look at. You can use unusual or colorful symbols on your graph. You can represent large numbers clearly with pictographs.

Pictographs are not often not very accurate. You must round data to represent it as pictures. People reading pictographs must be able to do mental arithmetic easily. You can not put a lot of categories of data on a pictograph.

Bar graphs

Bar graphs are more accurate than pictographs. The scale on a bar graph makes it easy to read the value of the bar. A large number of bars can be put on one bar graph.

Bar graphs are limited. Bars usually represent a category of data such as a data for a person or day of the week. If a bar graph is too "wide" then it is hard to read the value of the bars on the right.

Line Graphs

Line graphs can be very accurate. A lot of data can be put on a line graph. Scales can be put on both axes. It is easy to read a line graph because there are usually two scales. These graphs are good for representing data in a math or science class.

But line graphs are quite mathematical. It is hard to represent categories of data. You can't look at a line graph quickly and see what an individual data point means.

Circle Graphs

Circle graphs are good for representing data that is given in percents. Circle graphs are easy to understand quickly. They are good for representing data about money and budgets.

But circle graphs cannot be used when there is a lot of data. If there are too many categories, they are hard to read. Circle graphs only have one scale. They are not good for representing data from scientific experiments.

Summary

The first step in creating a graph is collecting data. Then you should think about what kind of graph to use.

- Should the graph be very accurate?
- Do you need to be able to read the graph at a glance?
- How many categories of data are there? Will they all be clear on the kind of graph you pick?
- Who will be reading the data? What kind of graph will they understand best?

Choosing the right kind of graph is important. It makes the data you represent much more readable.

Put the Future of Education in Your Hands...

Edutrends 2010
Restructuring, Technology, and the Future of Education
Education, Technology, and Paradigms of Change for the 21st Century
David D. Thornburg, Ph.D.

Pick up David Thornburg's visionary books that explore the social and technological trends shaping education.

Edutrends 2010—Restructuring, Technology, and the Future of Education

Thornburg examines the recent flurry of teacher bashing, moving from the reasoning flaws of those who would dismantle U.S. public education to the change process necessary for educational restructuring for the 21st century.

He proposes a way to think intelligently about the future while we reshape education to meet the needs of our society.

Education, Technology, and Paradigms of Change for the 21st Century

Learn unique patterns of thinking that encourage students to go above and beyond the three Rs. The technologies of calculators, computers, laser discs, and video games converge with new concepts of learning and intelligence to provide unprecedented opportunities for all students. Let this progressive book lead you toward a future filled with new paradigms.

International Society for Technology in Education
1787 Agate Street, Eugene, OR 97403-1923
Order Desk: 800/336-5191 Fax: 503/346-5890

Make This Step Your Next Step.

You've got your new Mac... Now what?

You want to know what those cute icons mean. You need to find out how to use that mouse-thing-a-ma-jig.

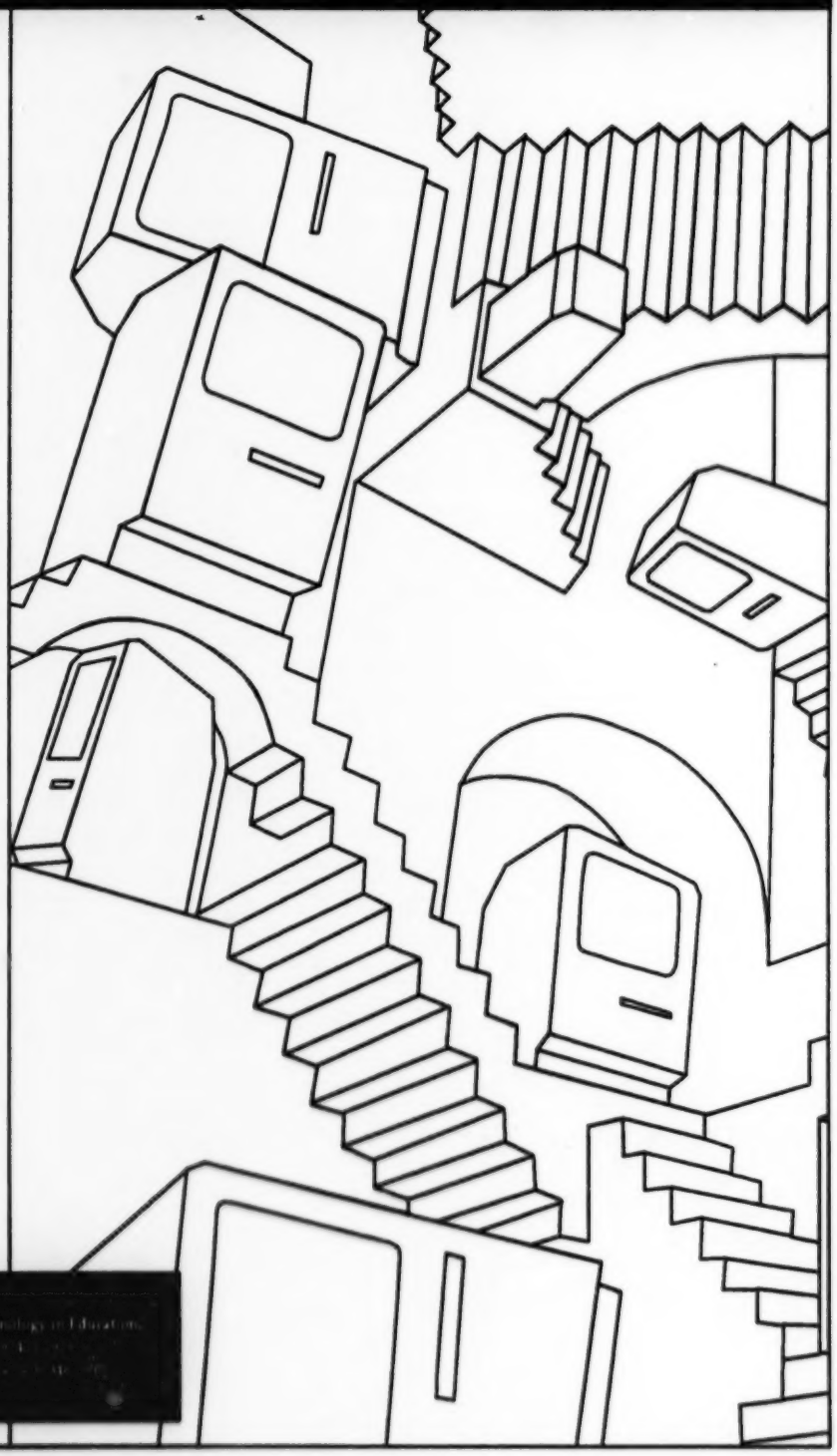
You'd like to know how to "empty the trash," and it'd be great to understand exactly what the "chooser" chooses.

By following the footsteps in ISTE's new book, *Macintosh Step by Step*, you'll pick up on all these basic Macintosh operating procedures, plus more—from finding the "power" button to demystifying the little Apple menu in the upper left corner.

And it also teaches you the theory behind Mac file storage and the importance of backing up your data on a regular basis.

In all, you'll find that this simple, illustrated guide will lead you on your way toward becoming a Mac user—not just a Mac owner.

Just call today and make your next computing step a step in the ISTE direction.



All you need is the right tool...

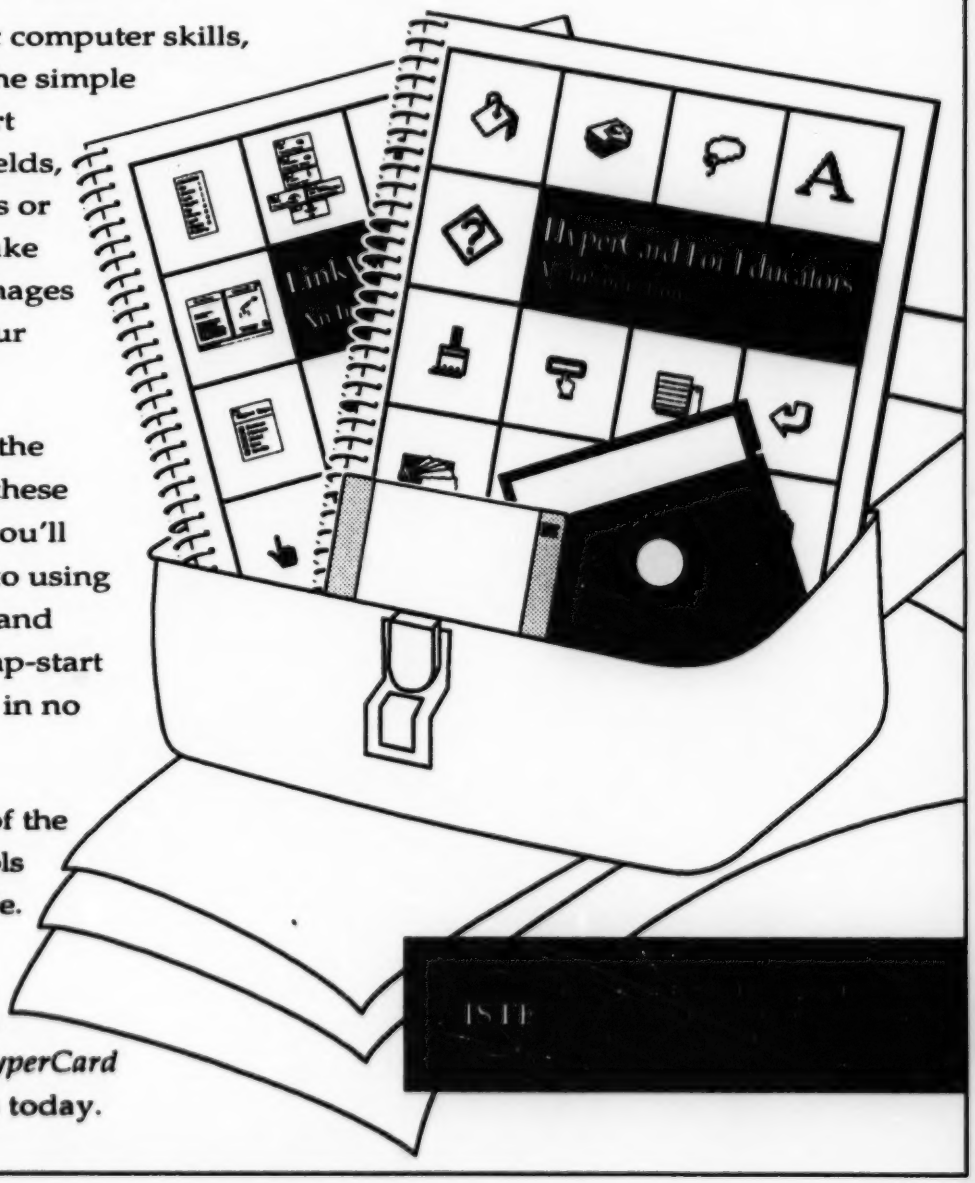
With either *HyperCard for Educators* (Macintosh) or *LinkWay for Educators* (MS-DOS) as an introduction to the interactive world of hypermedia, you can learn to build libraries of knowledge suitable for both classroom and personal use.

If you have basic computer skills, you can follow the simple tutorials and start using buttons, fields, and backgrounds or base pages to make your data and images come alive in your classroom.

By starting with the foundation that these books provide, you'll be on your way to using sounds, images, and animation to jump-start your curriculum in no time!

Take advantage of the organizational tools computers provide.

Call ISTE to order and begin building your *HyperCard* or *LinkWay* skills today.



Dave Moursund knows...

ISTE Executive Officer Dave Moursund is dedicated to improving education.

Since he first began working with computers in education in 1963, Dave has constantly sought to improve education at all levels through the use of appropriate computer-related technology.

As a continuing part of this effort, Dave's newest book—*The Technology Coordinator*—discusses the goals of education and how teachers and teachers of teachers can join in the enormous change process facing today's schools.

This book was written for educators who want to play a leadership role in the instructional use of technology. Dave uses interviews with technology coordinators to illustrate the needs, roles, and qualifications needed to accomplish the goals of educational technology.

In recent years, Dave has focused much of his writing efforts on problem solving and on staff development. Some of Dave's other books also available from ISTE include *Problem-Solving Models for Computer Literacy, LogoWriter for Educators: A Problem Solving Approach, Logo PLUS for Educators: A Problem Solving Approach* (co-authored with Sharon Yoder), and *Effective Inservice for Integrating Computer-as-Tool into the Curriculum.*

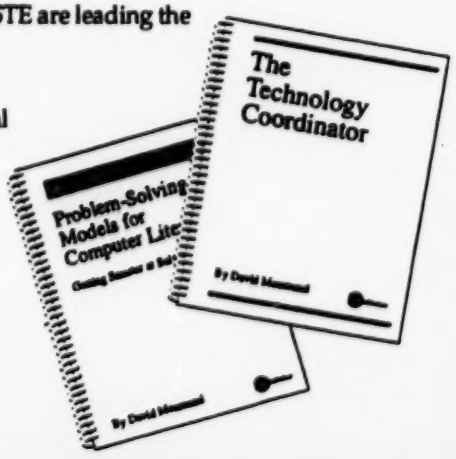
"Our educational system is not nearly as good as it could be," says Dave. "We know that computer-related technology can make substantial contributions to improving both the content of the curriculum and the instructional delivery system. Such technology can empower students and teachers."

ISTE believes that computer-related technology must become a tool that students and teachers use routinely if students are to be adequately prepared for adult citizenship in our Information Age society.

ISTE's wide range of periodicals, publications, projects, and other activities mirrors this vision and commitment.

Call us today to order Dave's latest books, and discover how he and ISTE are leading the way for the educators and school systems of tomorrow.

ISTE. Proudly putting you in touch with the world of educational technology.



International Society for Technology in Education
1787 Agate Street, Eugene, OR 97403-1923
Order Desk: 800/336-5191 Fax: 503/346-5890

BEST COPY AVAILABLE

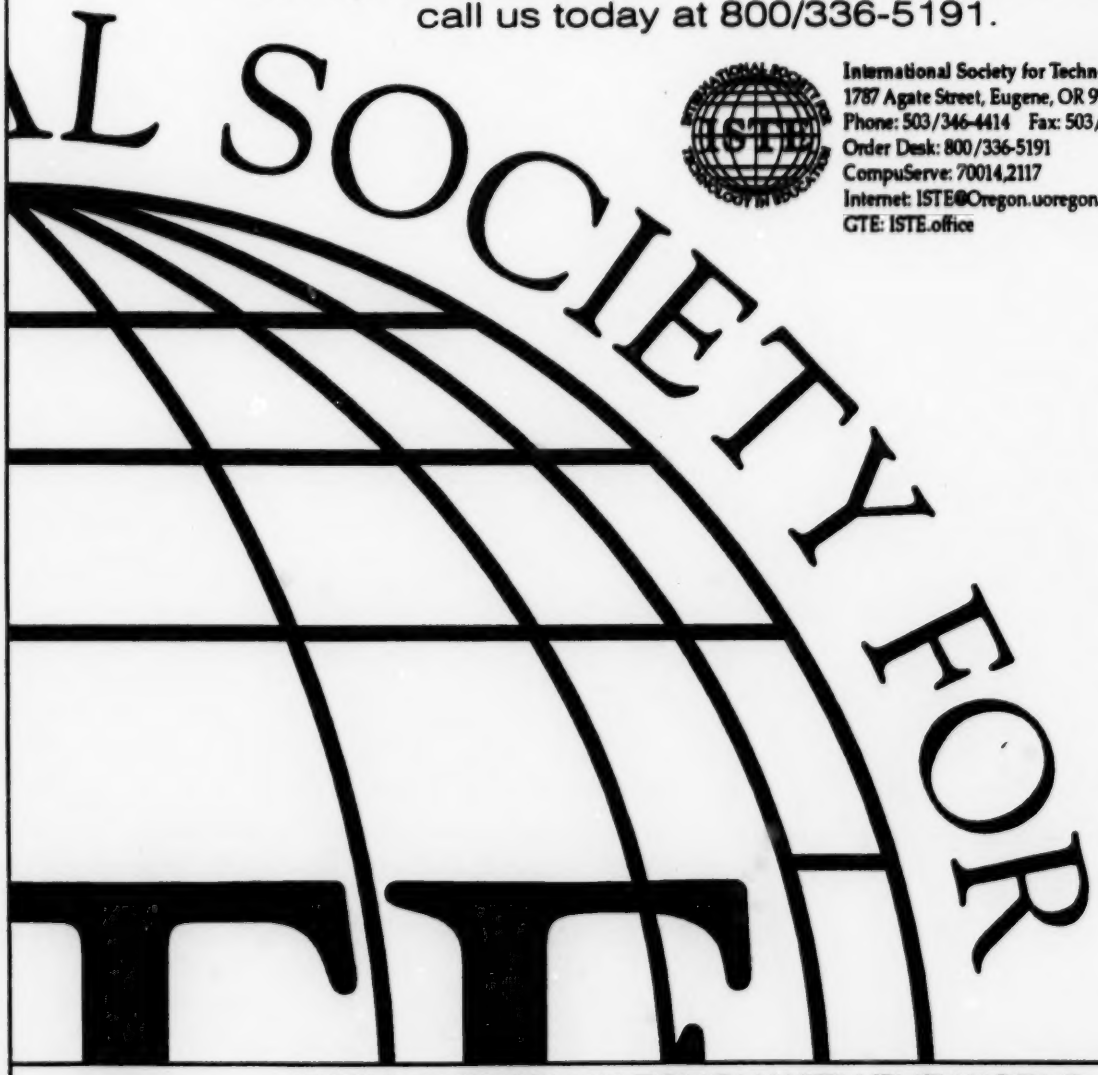
ISTE BRINGS THE WORLD OF TECHNOLOGY CLOSER TO YOU.

By drawing from the resources of committed professionals worldwide, ISTE provides support that helps educators like yourself prepare for the future of education.

ISTE members benefit from the wide variety of publications, specialized courseware, and professional organizations available to them.

They also can enjoy exciting conferences, global peer networking, and mind-expanding independent study courses.

So if you're interested in the education of tomorrow, call us today at 800/336-5191.



International Society for Technology in Education
1787 Agate Street, Eugene, OR 97403-1923
Phone: 503/346-4414 Fax: 503/346-5890
Order Desk: 800/336-5191
CompuServe: 70014,2117
Internet: ISTE@Oregon.uoregon.edu
GTE: ISTE.office

WE'LL PUT YOU IN TOUCH WITH THE WORLD.

BEST COPY AVAILABLE